Real-Time Trajectory Generation for Autonomous Nonlinear Flight Systems

AF02T002 Phase II Final Report
Contract No. FA9550-04-C-0032

Principal Investigators

Michael Larsen
Information Systems Laboratory Inc.
10070 Barnes Canyon Road
San Diego, CA 92121
voice: (858) 535-9680
fax: (858) 535-9848
email: mlarsen@islinc.com

Randal W. Beard
Department of Electrical and Computer Engineering
Brigham Young University
Provo, Utah 84604 USA
voice: (801) 422-8392
fax: (801) 422-0201
email: beard@ee.byu.edu

Timothy W. McLain
Department of Mechanical Engineering
Brigham Young University
Provo, Utah 84604 USA
voice: (801) 422-6537
fax: (801) 422-0516
email: mclain@byu.edu

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 074-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE 10 July 2006	3. REPORT TYPE AND DATES COVERED Final Report for 14 April 2004-14 April 2006

Real-Time Trajectory Generation for Autonomous Nonlinear Flight Systems

5. FUNDING NUMBERS FA9550-04-C-0032

6. AUTHOR(S) Michael L. Larsen Randal W. Beard Timothy McLain

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) INFORMATION SYSTEMS LABORATORIES, INC. 10070 Barnes Canyon Road. San Diego, CA BRIGHAM YOUNG UNIVERSITY Provo, Utah 84602

8. PERFORMING ORGANIZATION REPORT NUMBER ISL-3519-000-TR-06-01

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

USAF, AFRL
AF OFFICE OF SCIENTIFIC RESEARCH
4015 WILSON BLVD ROOM 713
ARLINGTON VA 22203

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

AFRL-SR-AR-TR-06-0361

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

A

13. ABSTRACT ***(Maximum 200 Words)*** Unmanned aerial vehicle and smart munition systems need robust, real-time path generation and guidance systems to avoid terrain obstructions, navigate in hazardous weather conditions, and react to mobile threats such as radar, jammers, and unfriendly aircraft. In Phase~I of this STTR project, real-time path planning and trajectory generation techniques for two-dimensional flight were developed and demonstrated in software simulation. In Phase~II these algorithms were refined, extended and were demonstrated in flight on a unique low-cost micro-air vehicle with a payload which included three optical flow sensors, a laser ranger, and a video camera. This report is a comprehensive technical summary of the significant work completed in Phase II to enable development of autonomous flight control systems which are capable of accomplishing the complex task of path and trajectory planning in dynamic and uncertain environments.

14. SUBJECT TERMS STTR Report, UAV, trajectory generation, way point path planning	15. NUMBER OF PAGES
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	

Executive Summary

Unmanned aerial vehicle and smart munition systems need robust, real-time path generation and guidance systems to avoid terrain obstructions, navigate in hazardous weather conditions, and react to mobile threats such as radar, jammers, and unfriendly aircraft. In Phase I of this STTR project, real-time path planning and trajectory generation techniques for two-dimensional flight were developed and demonstrated in software simulation. In Phase II these algorithms were refined, extended and were demonstrated in flight on a unique low-cost micro-air vehicle with a payload which included three optical flow sensors, a laser ranger, and a video camera. This report is a comprehensive technical summary of the significant work completed in Phase II to enable development of autonomous flight control systems which are capable of accomplishing the complex task of path and trajectory planning in dynamic and uncertain environments.

During Phase II, algorithms for both deliberative path planning, which assumes *a priori* knowledge of the environment such as a terrain map, and reactive path planning techniques, which are used to avoid unanticipated obstacles, were developed, simulated, and tested on a small UAV. Three dimensional path planning methods were developed which employ rapidly-exploring random trees (RRT) and genetic algorithms which efficiently find complicated paths through urban terrain. These path planners produce straight line segments which are tracked using a robust path following algorithm.

Reactive path planning techniques for obstacle avoidance algorithms were developed and tested extensively in both simulation and flight tests. These algorithms use optic flow and laser rangefinder measurements to sense obstacles and modify the waypoint path to cause the UAV to maneuver around the obstruction. As part of this effort, a light-weight optic flow sensor was developed and used for autonomous landing, and terrain and canyon following. The algorithm and sensor developed were tested in collision avoidance and canyon-following experiments.

Robust algorithms for accurate trajectory following in the presence of environmental disturbances were developed and demonstrated. A vector field path following concept which guarantees asymptotic tracking in the present of constant wind was implemented and shown to reject disturbances up to 70% of airspeed.

Contents

1 Phase II Technical Objectives

Main Objective. Demonstrate real-time autonomous path planning and trajectory generation techniques using limited computational resources on small UAV hardware. This main objective was supported by the following technical objectives.

Objective 1. Extend the path planning and trajectory generation algorithms to three dimensions. Low-altitude flight, a critical capability for agile munitions with low-resolution sensors, requires the UAV to track the terrain and fly over threats.

Objective 2. Robustify the trajectory generation algorithms against wind and other environmental disturbances. The trajectory generation algorithm demonstrated in Phase I was to be adapted to account for wind and other environmental disturbances. In Phase II we were to modify the trajectory generator to include feedback from the tracking error.

Objective 3. Demonstrate agile autonomous flight on small UAV hardware. A hardware-in-the-loop demonstration of real-time path planning and trajectory generation algorithms of Phase I and modifications developed in Phase II was to be performed with small UAVs. All algorithms were to be flight tested using a virtual sensor environment using simulated height-above-ground sensors.

Objective 4. Develop and integrate a virtual sensor environment to allow hardware-in-the-loop demonstration of agile trajectory generation. To accomplish this, we set out develop a virtual sensor environment that will use UAV GPS location, altitude, and topological maps to provide simulated height-above-ground information to the UAV.

2 Phase II Work Accomplishments

In this section, we describe the progress made toward accomplishing the Phase II technical objectives. During the first year of the project a virtual sensor environment was developed and flight tested the results as planned. During the second year of the project, we investigated several sensors for collision avoidance and flight tested the associated algorithms developed. In this report, we focus on the actual sensors and their performance rather than the virtual sensor environment. In Section 2.1, we describe the approach to 3D trajectory generation developed in Phase II for both deliberative and reactive path planning. In Section 2.2, we report results on a novel path following technique that compensates for significant wind disturbances. In Section 2.3 we describe sensors and algorithms developed for collision avoidance. In Section 2.4, we summarize flight test results.

2.1 Objective 1: Deliberative and Reactive Path Planning

Path planning in constrained urban terrain where maneuverability is limited is a challenging problem. We have developed path planning algorithms for small fixed-wing unmanned air vehicles. We explored several path planning approaches in Phase II including Rapidly-exploring Random Trees (RRTs) and genetic algorithms which will be described in Sections 2.1.1 and 2.1.2, respectively. Both of these methods generate waypoint paths using *a priori* knowledge of a terrain map and are classified as deliberative path planning techniques. We have also investigated reactive path planning techniques that enable collision avoidance when obstacles are encountered in real-time. Collision avoidance sensors and algorithms will be discussed in Section 2.3

2.1.1 Deliberative Path Planning using RRT Algorithms

One approach developed for trajectory generation uses the RRT algorithm and extends the work of Lavalle and Kuffner [1]. This modified RRT algorithm is able to quickly create feasible UAV waypoint paths. Portions of this work were published in [2]. The task of the a planner is to find a traversable path from a starting position x_{start} to a goal position x_{goal} through a terrain, or configuration space $\mathbb{C}$. To solve this problem for micro air vehicles (MAVs) in urban terrain, we developed a three dimensional planner that can handle nonholonomic vehicle constraints and tight turning requirements. In [3], Lavalle introduced Rapidly-Exploring Random Trees (RRTs) as a potential solution to such path planning problems. Important advantages of this method over other techniques are discussed in [4].

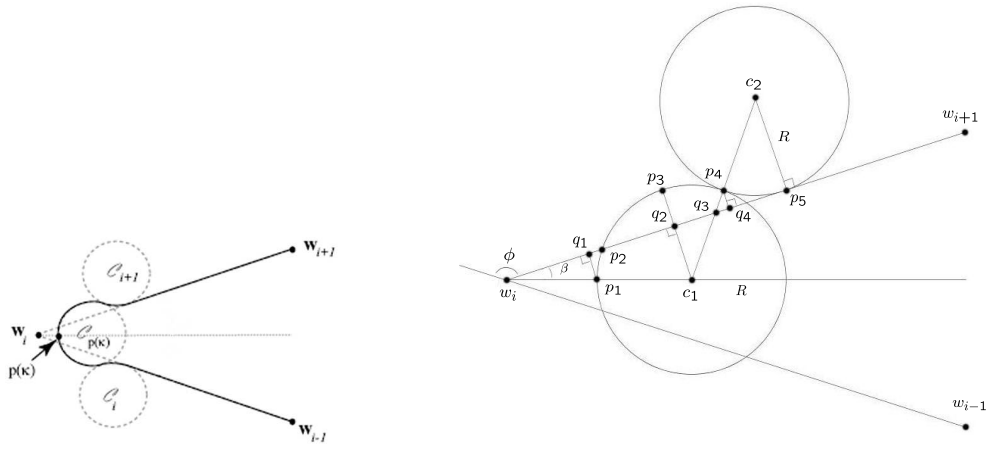
The basic RRT algorithm is as follows:

1. Pick a random state x_{rand} in $\mathbb{C}$.
2. Using a metric ρ, determine the node x_{near} in the tree that is nearest x_{rand}.
3. Apply a control input u to move the graph toward x_{rand} an incremental distance.
4. If there are no collisions along this segment, add this new node x_{extend} to the tree.
5. Repeat until you have reached x_{goal}.

We have implemented a version of the RRT algorithm to plan 3D waypoint paths for MAVs, and have successfully used the algorithm in both simulation and flight tests. The RRT algorithm that we implemented differs in some key ways from the basic algorithm listed above. The standard RRT algorithm produces a time-parameterized set of control inputs to move from x_{start} to x_{goal}. The validity of this result depends on the accuracy of the state-space model being used. In real-life situations, we encounter sensor inaccuracies,

wind, and other unmodeled factors which limit the performance of open-loop path planners. We have combined RRTs with closed-loop path planning. Planned paths are represented as waypoints and the paths are tracked by an autopilot using trajectory smoothing [6, 7, 8]. Knowing the feedback control policy used by the autopilot, and therefore the expected cross-track error at points along the path, we can plan valid paths without specifying the exact control inputs.

The most significant kinematic constraint on MAVs is their inability to instantaneously change direction. In an urban environment, we would of course want to be able to turn sharply. However, since this is not possible, planned paths must take into account these limitations on the MAV. Quantifying how close the MAV will stay to the waypoint path provides useful information that can be utilized by the RRT path planner. Important previous research on constrained trajectory generation is found in [6, 7, 8]. Kingston shows in [8] that due to control limits, the MAV has a limited local reachability region, bordered by minimum radius circles. In [6], Anderson proved that a time-optimal trajectory along a waypoint path will be a sequence of straight-line path segments combined with arcs along these minimum radius circles, properly placed near the vertex. Anderson showed that a class of trajectories called κ-trajectories are time-extremal.



(a) Waypoint path smoothing with κ-trajectories (from [8]).

(b) Definitions of segments.

Figure 1: κ-trajectories.

Definition 2.1 *A κ-trajectory is defined as the trajectory that is constructed by following the line segment $\overline{w_{i-1}w_i}$ until intersecting C_i, which is followed until $C_{p(\kappa)}$ is intersected, which is followed until intersecting C_{i+1}, which is followed until the line segment $\overline{w_iw_{i+1}}$ is intersected, as shown in Figure 1(a).*

These κ-trajectories always pass over $p(\kappa)$. Depending on the situation, different values of κ can be chosen. $\kappa = 0$ will guarantee a flight over the waypoint. $\kappa = 1$ will guarantee a minimum-length path by cutting each corner. Anderson also shows in [6] that κ can be chosen to guarantee that the trajectory will be the same length as the waypoint path.

Using this information about the control law used by the MAV for trajectory generation, we can plan a waypoint path that can be tracked while avoiding collisions with the terrain. Referring to Figure 1(b), we can visualize the MAV moving along the segments from w_{i-1} to w_i to w_{i+1} where the 2-dimensional turn angle at w_i is ϕ and estimate how far we

Segment on Path	Maximum Inside Distance	Maximum Outside Distance
$\overline{w_i q_1}$	0	0
$\overline{q_1 p_2}$	$\overline{p_1 q_1}$	0
$\overline{p_2 q_4}$	0	$\overline{p_3 q_2}$
$\overline{q_4 p_5}$	0	$\overline{p_4 q_4}$

Table 1: Distances from Path to Trajectory

expect the MAV to vary from the straight-line path at any given point along the trajectory. Considering the straight line segment $\overline{w_i w_{i+1}}$, the optimal trajectory starts at p_1 ($p(\kappa)$), follows circle 1 through p_2 and p_3 to p_4, switches to circle 2 and follows it to p_5, then follows $\overline{w_i w_{i+1}}$ toward w_{i+1}. Using symmetry, we can restrict attention to one half of the path. We divide $\overline{w_i w_{i+1}}$ into smaller segments, and in each segment we know the maximum perpendicular distance the trajectory will be from $\overline{w_i w_{i+1}}$ (both inside and outside the path). The segments representing these maximum distances are listed in Table 1.

From [7] we know that

$$\overline{w_i p_1} = \kappa R \left(\frac{1}{\sin(\frac{180-\phi}{2})} - 1 \right) = \kappa R \left(\frac{1}{\cos(\frac{\phi}{2})} - 1 \right). \tag{1}$$

Using this information and basic geometry we can determine the remaining distances listed in Table 1. The distance information is used in the RRT algorithm to determine possible collisions with the terrain. Paths that present a collision threat are eliminated from the tree.

Path smoothing was used to improve the RTT output and reduce the number of waypoints the MAV must traverse. Nodes in the path planned by the RRT planner are tested to see if they are truly necessary or if they can be skipped, with a direct edge from a previous node to a future node. Because of the incremental and random growth of the RRT, resultant paths will not be straight for long distances, even in long corridors where a straight path will do. The path smoothing step eliminates these extraneous nodes, which are important in growing the tree, but which can be later deleted.

The RRT path planner has three different altitude modes: constant climbs, constant altitudes, and constant height above ground. The standard mode is where the MAV can change altitude, irrespective of the terrain. This is effective in urban terrain flying, where it may be more efficient to climb and fly over low buildings instead of planning a longer path around them. Incremental collision testing for this mode is done along a constant climbing path (within the maximum slope limits) from x_{near} to x_{extend}. To fly at a constant altitude we mapp x_{rand} to the constant altitude plane. We can also can generate paths at a constant height above ground. Here, the mapping is done from x_{rand} to a state at the same x and y position, but at the designated height above the terrain. Incremental testing in this case must test between the two points to make sure there are no sharp altitude changes that the MAV sensors would not be able to quickly compensate for. In this mode, the resultant paths are over relatively smooth segments of the terrain.

2.1.1.1 Modifications to the RRT algorithm that ensure collision avoidance

There are several constraints that must be considered when planning a path for a UAV. A UAV path typically consists of a series of waypoints that must be followed. Using a basic waypoint following algorithm, the UAV will be able to track this path from start to finish.

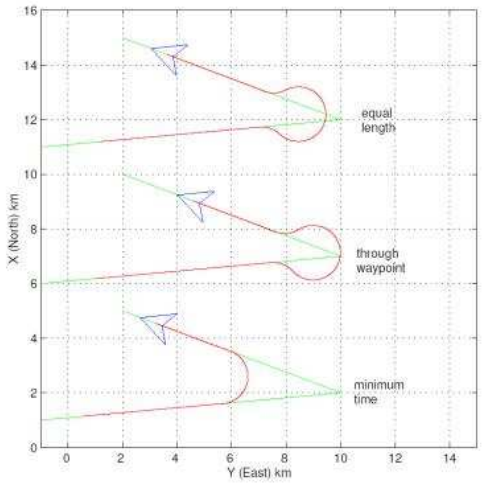


Figure 2: Possible trajectories created from the same waypoint path.

However, at each waypoint, the UAV is suddenly commanded to change direction. UAV dynamics limit the speed of this response, and the path cannot be tracked perfectly at each turn.

Different trajectory generation methods can be used to smooth waypoint paths and provide a trajectory without discontinuities in course rate. These methods typically involve the addition of circular orbits at the turns, as seen in Figure 2. Trajectory generation allows us to know the exact path the UAV will follow, but it makes waypoint path planning more difficult. We cannot just guarantee that the waypoint path is free from collisions; we must verify that the actual trajectory will not cause a crash. Figure 3 shows a situation where a planned waypoint path does not collide with any buildings, but the smoothed path does.

In path planning, we must be able to quantify how far the trajectory will take us from our waypoint path. With this information, we can perform the proper collision testing and ensure that the UAV will not crash. The distance that the trajectory will be from the path is a function of the radius of the orbit, the angle of the turn, and the distance along the path from the vertex. The geometric calculations are given in the previous section. The radius of the orbit is determined by the minimum turning radius of the UAV. We use a conservative estimate of 50m, knowing that we can track such orbits even in extreme wind conditions. Naturally, as the turns become sharper, the deviation from the path will increase. We had to place limitations on the sharpness of the turns and the distance between waypoints, because if two sharp turns are very close together, one turn will not finish before the next begins. With these constraints considered, we are able to test any path and determine whether or not a UAV can follow it, as well as determining if there will be a collision with the terrain. This collision testing will be an important part of our algorithm.

Rapidly-exploring random trees were created as a motion planning method in large dimensions with dynamic constraints considered. The algorithm works by starting from a given state and growing a tree (see Figure 4) of reachable states toward a final state. The process of growing this tree involves picking a random state in the configuration space, choosing the current state that is closest to that state (by a predetermined metric), then applying a control input that will pull the chosen state toward the random state. The new path is tested for collisions, and if none are found, then the input and state are added to the tree. This process is continued until the goal state is reached. The path from the start

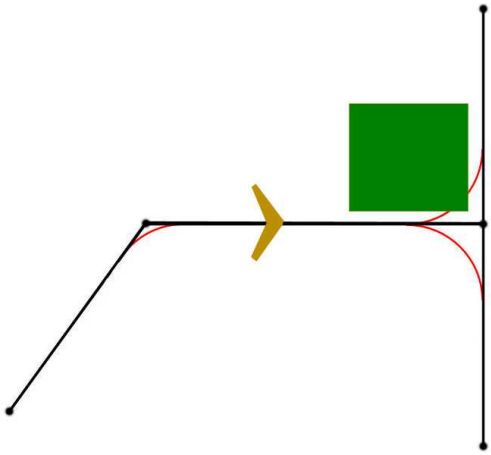


Figure 3: A collision with the building is caused by the trajectory generator.

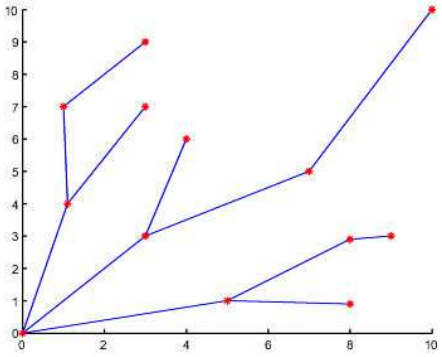


Figure 4: An example of a growing tree.

to the finish is a series of intermediate states connected by control inputs. In effect, we have a time-parameterized series of control inputs to drive us through the configuration space.

RRTs have been proven to effectively solve problems that no other motion planning method can solve. Many other path planning methods involve laying out a grid of possible states, performing local planning between the states, then trying to find the shortest path from start to finish. This could be effective for UAVs in open areas, but not in urban terrain. One main reason is that in tight areas, it is difficult to plan a path between two points without having some information about the path immediately before and after. As discussed above, a waypoint path cannot be followed exactly, and the real trajectory depends on the previous segment. In open areas, this problem can be ignored, but it is a key part of planning in urban terrain.

We found that the underlying concepts of RRTs map very well to the domain of UAV path planning in urban terrain. One important concept is that of growing the tree of possible paths. Before adding a new branch, we can use the previous branch to predict the real trajectory and make sure our path is collision-free. The tree will therefore only contain flyable paths. RRTs have also been shown to quickly explore the configuration space. Therefore, we can quickly search the possible area and choose a path that we know

we can follow.

The output of an RRT path planner is a list of control inputs and the times when they should be applied. This is just an open-loop solution to the planning problem. This is effective in simulation, but not in real life where disturbances and model inaccuracies will prove the solution insufficient. RRTs are also very computationally intense when used in high dimension problems. The speed of our planner is important, even though this planning is done prior to flight. It may at times be necessary to quickly update the path, and in those cases, we prefer to have the planner find a good path in seconds rather than minutes. Also, RRTs are based on choosing random states to move toward. There is no guarantee of optimality, although the longer we take to plan, the more completely our area will be explored. We must work to overcome these drawbacks in order to produce an effective planner.

We have been able to use the conceptual advantages of the RRT algorithm and modify it so it will work with our UAVs in constrained areas such as urban terrain. First, we dealt with the problem of open-loop path planning. The RRT algorithm assumes knowledge of of the system dynamics, which is not available to us. Instead we use useful approximations and place limits on the commanded turn radius and climb rate. As long as our planner does no exceed these limits, we do not have to worry about the exact controls that will get us from point to point.

We therefore can do our path planning in the output space, instead of the input space. Instead of choosing the exact controls to get between states, we selected 3-D positions in our space and set these as our waypoints. We can connect these states with straight lines, then use the methods discussed earlier to determine the actual smooth trajectory the UAV will take when given that path. Our metric for choosing the closest state in the RRT algorithm therefore included tests for steepness of the path and sharpness of the angle, along with a test to make sure we could finish a turn before the next one began. Through this method, we were able to use the RRT idea of growing a tree, but deal with waypoints instead of open-loop controls.

We also biased the RRT algorithm toward the goal state by permitting that occasionally be the "random" state chosen. We also explored the option of either picking the first path that is found or searching for many paths and choosing the one of least cost. Since the RRT is a randomized planner, the initial path found may be very different from one planning session to the next. However, once an initial path is found, most future paths are very similar. Searching for more paths is sometimes effective, but usually the time spent looking for more paths is not very well spent. It is typically more effective to smooth the path by eliminating extraneous nodes, creating a new path of lower cost, but that still meets the constraints. Through this process, we can very quickly find a short path through the terrain with few waypoints.

The steps of the modified algorithm are as follows:

1. Pick a random point in 3-D. With small probability, set the goal position as the random point to pull the graph toward the goal.

2. Determine the node in the tree that is nearest the random point. The metric is defined as the closest point in Cartesian space, with constraints on the 2-D turn angle and the longitudinal slope to the new point.

3. Move an incremental distance from the chosen point toward the random point, resulting in a temporary state.

4. Randomly pick a maximum 2-D turn angle for leaving that segment, and extend past the temporary state far enough for a turn of that size to begin. This is our new state.

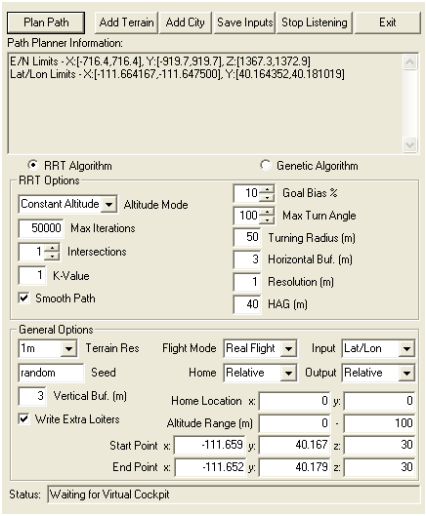


Figure 5: Path Planner Options.

5. Search along the path from the chosen state to the new state to see if there are any intersections with the terrain. Knowing the turn angle and how the path will be smoothed, verify that the real trajectory is also collision-free.

6. If no collision is found, add the new node and the edge to the tree.

7. Test to see if the new state can be connected directly to the goal. If so, you have a valid path from start to finish. If not, then return to step 1.

8. Continue until you have found as many paths as you are searching for. Pick the path of least cost and smooth the path.

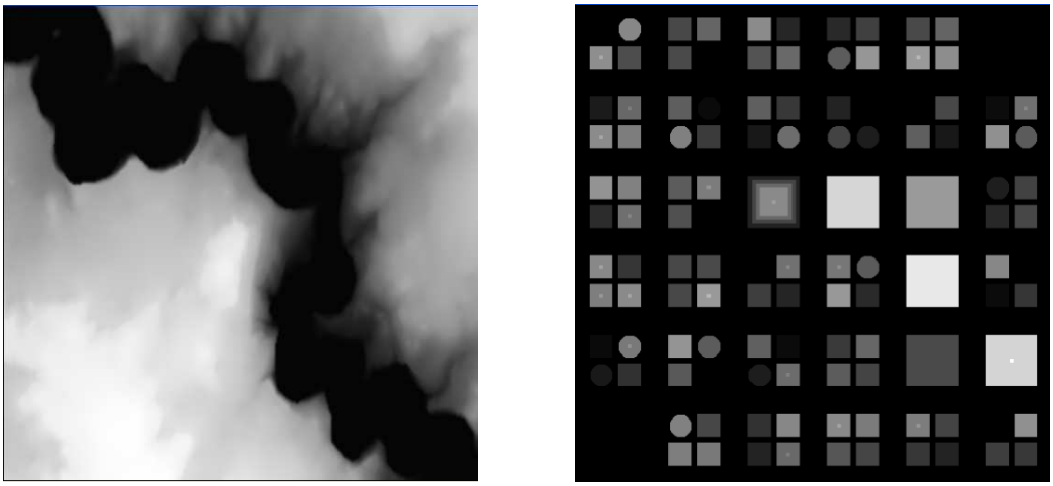
The RRT path planning algorithms developed in Phase II can be customized by the user depending on the UAV and the terrain. Figure 5 displays the options available to our path planning program, and below we also have a description of some of these options.

Maps Terrain data from the USGS can be downloaded and read by the path planner. A map of any area can be chosen to do the planning in. An example map of a narrow canyon is found in Figure 6(a).

Cities An xml city format was developed to represent urban areas. These can be overlayed on any terrain map. This format could be used to build a rough model of a real city and practice planning paths. An example city is found in Figure 6(b). Lighter buildings represent larger heights.

Start/End Locations Paths can be planned between two absolute locations (latitude and longitude) or between relative locations on a map, using relative UTM coordinates.

Altitude Mode The path planning can be done at a constant altitude, at a constant height above ground, or at changing altitudes with constant climb rate paths between waypoints. In constant altitude planning, we effectively take a slice of the terrain at that altitude and perform 2-D path planning. When planning at a constant height above ground, we tend to search for areas of the terrain without steep inclines. Planning with constant climb rate provides the option of flying up and over a building/obstacle



(a) A canyon map using USGS data.

(b) A virtual city.

Figure 6: Terrains used by the path planner.

instead of a longer distance around it. The minimum and maximum altitudes can be specified in any case.

Other We can also specify the number of paths to find, whether to smooth the paths, the amount of bias toward the goal, the maximum turn angle, the minimum turn radius, the step size in collision testing, and the horizontal and vertical buffer zones that a path must clear.

Simulation Results

Planning paths in urban terrain using the modified RRT algorithm has proven to be extremely effective. Here are presented some of the results that were obtained in Phase II. We used city models with realistic parameters in order to test the path planner. Figure 6(b) is one of these test cities. These cities were randomly generated, with input parameters including the range of building heights, building density, street width, and city size. Although other factors are included, these were the most significant in determining the speed of the planner and its ability to plan paths.

We first tested the planner in finding constant-altitude paths through this virtual city. Figure 7 shows a tree grown from one corner of the city to the opposite corner. We can see the RRT tree exploring some different streets until it find a clear path to the goal. Figure 8 includes plots of planned paths through a city at constant altitude, a different city at varying altitudes, and also through a narrow canyon. The path planner can handle a variety of terrains and quickly plan paths.

The RRT path planner was written in Visual C++ and experiments were conducted on a 2.4GHz Pentium 4 PC running Windows XP, with 512MB RAM. The average time it takes to plan a path through the city in Figure 8(a) is under .5s, but due to the randomness inherent in the algorithm, this time ranges from 15ms to 5s. The city in Figure 8(b) is actually a lot larger, but the path planner runs faster with that city. One main reason is that it has a few more open areas to fly in. The path planner quickly finds open areas, because those are the areas where it is easiest to maneuver in.

This is a general pattern found with all of the cities. As cities become more congested or the streets become narrower, computation time increases. In cities that don't have a full building on every block, the path planner can find paths very quickly, with street widths

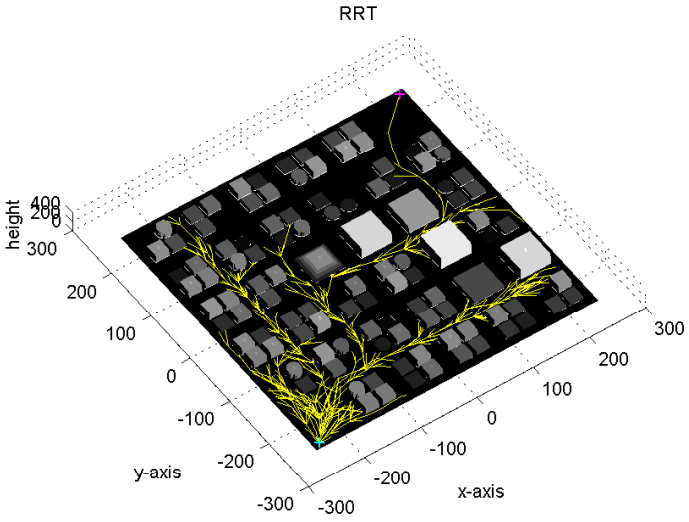
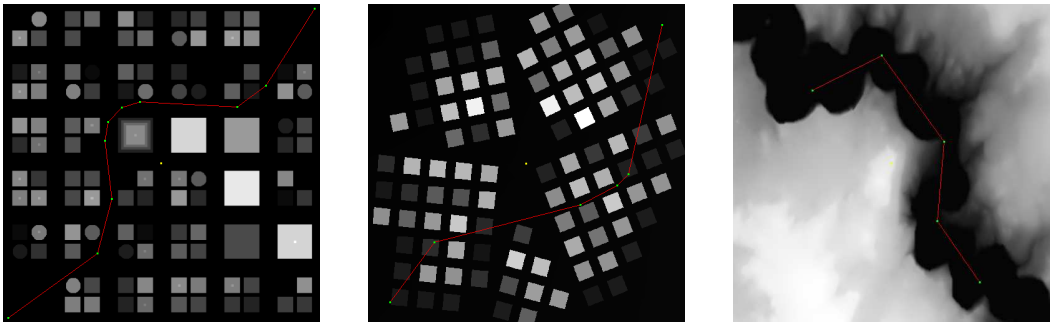


Figure 7: MATLAB plot of an RRT Tree.



(a) Path through a city at constant altitude.

(b) Path through a city with constant climbs.

(c) Path through a canyon.

Figure 8: Planned paths through different terrains.

even down to 15m. However, in cities with tall buildings occupying each block, the streets must be at least 30m wide to consistently find paths. When the path planner is allowed to change altitudes, the time to plan a path decreases because the UAV can fly over lower buildings. To fly through cities with narrow streets, this is often necessary.

The most important question is how effective this path planner would be in a realistic urban environment. These virtual cities have been based on estimates of the appropriate parameters. Typical cities have streets ranging from 15m to 30m wide, with 20m streets being typical. Our testing showed that it would take a large grid of tall buildings occupying many contiguous blocks, with less than 30m between them, to prevent the path planner from finding a path. Even cities with many tall buildings tend to have large amounts of space between the buildings. While many cities have narrow streets, they also have open areas and some lower buildings over which the UAV can turn. Testing has shown that this planner will tend to find these areas rather than making several difficult turns.

The answer to our question really depends on how high we want to fly relative to the average building height in the city. If every building was at least 40m high and we wanted to fly at 30m altitude, the streets would probably have to be 30m wide, otherwise the path planner would have trouble. In that situation, the UAV would be trying to make turns with a radius of 50m while the street width was only 30m. Sharp turns would be very difficult. However, if even only 80% of the buildings are at the altitude we want to fly, planning paths will become easy. If we permit the planner to change altitudes, planning also becomes easier.

It is important to note that this path planner is limited by the accuracy of the a-priori terrain map. If a path is planned with an inaccurate model, additional sensors may be required on the UAV so it can dynamically adjust its path to avoid unmodeled obstacles. If the map is precise, though, the path planner has proven to be extremely fast, and the UAV will be able to fly the exact path without collisions. This RRT algorithm is very fast, and would certainly meet any speed requirements for an a-priori planner. Because of the randomness, optimality is not guaranteed, but the algorithm will still quickly find a flyable, collision-free, and low cost waypoint path.

We have used a simulator to test how well this planner works. Our terrain maps and city files can be imported into this simulator to give us a visual image that verifies that we are not colliding with any obstacles. When commanding these planned paths to the UAV in simulation, we can fly through any terrain that we are able to plan a path through. The path planner accounts for many of the characteristics of the UAV flight, such as turning radius and path following mode, and also adds in some buffer area just to be safe. The resulting paths end up not coming very close to the obstacles, so small biases or deviations from the path do not lead to crashes. These simulation results verify that the path planner is producing traversable paths. Figure 9 shows some screenshots of the UAV flying a path through a city.

2.1.2 Deliberative Path Planning using Genetic Algorithms

The path planning task can be viewed as the generation of a waypoint path starting at the UAV's location and ending at the UAV's goal location. Multiple algorithms have been applied to path planning for robots in the presence of known obstacles. Some of these algorithms are potential fields, voronoi graphs, rapidly exploring random trees (RRT), and the famous genetic algorithm. We implemented a genetic algorithm that plans a path for a UAV in urban terrain in Phase II. We assume that we have access to a map containing the altitude of any point on the terrain including the altitude of any buildings. The genetic algorithm successfully creates a waypoint path that allows the UAV to navigate safely through simulated cities. Additionally, the paths conserve UAV power usage by minimizing

(a) UAV enters a virtual city.

(b) UAV turns down a street in the city.

Figure 9: Screenshots of a flight in a virtual city.

distance while avoiding obstacles.

2.1.2.1 Genetic Algorithm Setup Our goal is to get from a start point to an end point while avoiding obstacles, such as mountainous terrain or tall buildings. A genetic algorithm can be used to solve this problem. We define our genetic algorithm population as a set of N paths. Each path is composed of n waypoints. The algorithm is comprised of the following steps

1. Generation of Initial Population
2. Tournament
3. Crossover
4. Waypoint Mutation
5. Path Mutation
6. Elitism
7. Fitness.

This genetic algorithm also uses the following parameters

- Number of generations
- Population size
- Tournament size
- Waypoint mutation probability
- Maximum waypoint mutation
- Waypoint addition probability
- Waypoint deletion probability
- Alpha (α) - used to generate initial population
- Initial number of waypoints per path.

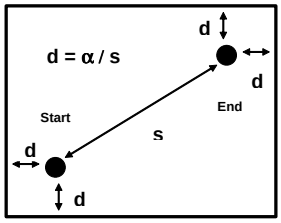


Figure 10: The region of possible waypoints of the initial population.

These parameters are discussed in the following sections.

Generation of Initial Population Initially, waypoints are randomly chosen from a bounding region that encompasses the start and end points. The bounds of the region are defined by a dimensionless variable α that determines the distance between the start and end points, and the north and east coordinates of the bounding region as show in figure 10. The actual distance d is defined by $d = \frac{\alpha}{s}$ where s is the distance between start and end points. α is defined in this way to make the algorithm easily scalable without having to retune the parameters. A relatively large α will allow for initial paths that explore the area more fully while small α create initial population that are shorter and more goal oriented. Ideally, a small α will be used, but in the presence of many obstacles, a larger α may be beneficial. The user is free to specify the value of α so that the UAV may navigate a long ways from the straight line path when necessary. Normally, it won't be necessary to plan paths that go outside this region. Additionally, mutation allows the path to mutate outside this box which provides a more complete design space search while still using a small α.

The altitude range for the waypoints is also defined initially by the user. Waypoints will not vary from this altitude range. The number of waypoints initially in the path is also a parameter the user can specify. Once waypoints are created for a path, they are ordered based on distance from the start point. Such an ordering may result in a zigzag like pattern which would most likely be suboptimal. If a large value of α is used, the initial paths will not only zigzag left and right from the path, but forward and backward because some initial waypoints may be behind the start point. However, such paths will have a low fitness and more optimal paths will be found after successive generations of the genetic algorithm.

Once the initial population is created, the genetic algorithm runs the number of generations specified by the user. The steps in each generation are listed below.

Tournament Tournament selection was used to choose which parents to use for breeding. Multiple population members are selected randomly, and the member with the best fitness is chosen as a parent. The user may select the size of the tournament. A low tournament size will result in greater variation in the population. As the tournament size is increased, more paths compete to be a parent and genetic variation is reduced.

Crossover Once two parents have been chosen, path crossover is performed at a random distance along each path. The distance is chosen to be shorter than the shortest path and is the same for both parents. The parents are divided between the two path waypoints closest to the crossover point creating two path segments for each parent. One path segment from each parent is traded with a corresponding segment from the other parent forming two new paths.

Waypoint Mutation.

After crossover, the waypoints of the children are mutated. The mutation function iterates through each waypoint in a path and randomly decides whether to mutate that point. The probability of mutation is specified by the user. If mutation occurs, the point is moved randomly within a ball around its current location. The size of the ball (maximum

mutation) is also specified by the user.

Path Mutation. The mutation function may also randomly add or delete a waypoint in a path. Waypoint addition and deletion probabilities are set by the user. For waypoint deletion, a waypoint is chosen randomly. For waypoint addition, a waypoint is randomly generated and added into the path between the two waypoints in the path which are closest to the new waypoint. Both waypoint and path mutation are very helpful in introducing genetic variation into the population. If a path from the initial population is infeasible (intersects the terrain or buildings), mutation can modify or create points that bypass the obstacle.

Fitness The fitness of the path is determined using the length of the path (s), the vertical climb (V_c), the descent (V_d), and the number of path segments that intersect the terrain (i) as defined by

$$f = (s + k_c V_c - k_d V_d)(1 + p \cdot i). \tag{2}$$

k_c is the climb factor, k_d is the descent factor and p is the penalty for a terrain intersection. This function was minimized to provide the path of least distance and fewest terrain collisions. For the experiments shown in this document we used $k_c = 1$, $k_d = 0.5$, $p = 2$. This function works well for path planning because it rewards paths of shorter length while assigning a high penalty to terrain collisions. This may not be a good fitness function in the event that there is a very large obstacle that is difficult to fly over or around but flying straight through is a very short path. If the cost to fly around or over the obstacle is $(p+1)$ times as much as flying straight through, the path with the collision will have a better fitness. Increasing p decreases the possibility of accepting a terrain collision.

Elitism After crossover and mutation, children are added to the population, although they are not valid options for tournament selection until the next generation. The $2N$ paths in the population are then ordered by fitness and the paths with the best fitness are chosen to propagate to the next generation.

2.1.2.2 Simulation Testing

The effectiveness of the genetic algorithm path planner is sensitive to the parameter values set by the user. We tested the algorithm to find default parameter values (see table 2) that would result in a feasible path most of the time. We started our test path in the middle of a virtual city, surrounded by tall buildings. The end point was situated in a corner of the map outside of the city, requiring the UAV to navigate around buildings. Once the default parameters were set, we tested high and low values for each of the parameters (see tables 3 and 4), to see which had the most effect on optimization. During the analysis the same random numbers were used and only the tested parameter was varied from the default. Best fitness and elapsed time for each of the varied parameters is discussed in section 2.1.2.3.

2.1.2.3 Figure discussion

In the following figures, having low fitness value means the path is shorter and therefore better. The bar charts indicate the time to run the algorithm for 30 generations on an IBM thinkpad with a 1.86GHz processor and 512MB of RAM.

Figure 11 shows the effect of the number of generations on the genetic algorithm. All of the data sets have the same fitness history because we used the same random numbers. The data set for the low value is underneath the solid line and ends at 15 generations. The best fitness appears to decay exponentially with the number of generations. Also, the computation time increases linearly with the number of generations.

Figure 12 shows population size analysis. Large populations produce a path with good fitness faster, but increase computation time.

Figure 13 shows the effect of tournament size. A Large tournament size produces a path with good fitness more rapidly but may cause the population to converge to a local

Parameter	Value
Number of Generations	30
Population Size	150
Tournament Size	3
Waypoint Mutation Probability	0.3
Maximum Waypoint Mutation	60
Waypoint Addition Probability	0.2
Waypoint Deletion Probability	0.2
α	0.05
Initial Number of Waypoints per Path	30

Table 2: Default user parameter values for the optimization

Parameter	Value
Number of Generations	15
Population Size	100
Tournament Size	1
Waypoint Mutation Probability	0.1
Maximum Waypoint Mutation	20
Waypoint Addition Probability	0.05
Waypoint Deletion Probability	0.05
α	0.01
Initial Number of Waypoints per Path	15

Table 3: Low user Parameter values for the parameter analysis

Parameter	Value
Number of Generations	50
Population Size	200
Tournament Size	5
Waypoint Mutation Probability	0.5
Maximum Waypoint Mutation	100
Waypoint Addition Probability	0.4
Waypoint Deletion Probability	0.4
α	0.2
Initial Number of Waypoints per Path	50

Table 4: High user parameter values for the parameter analysis

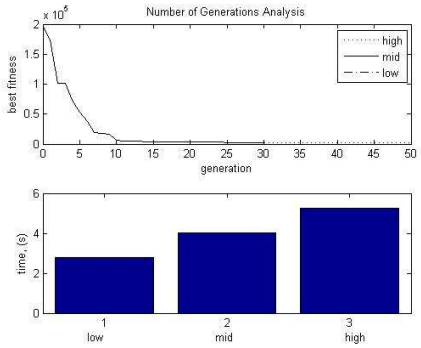


Figure 11: Best fitness and elapsed time are shown for the low, mid, and high values of the number of generations.

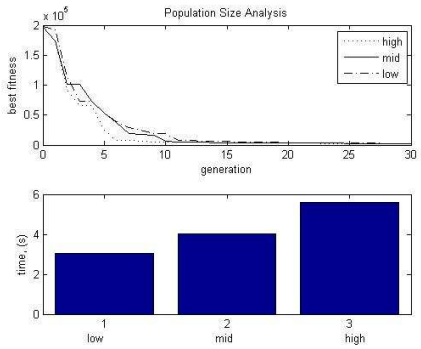


Figure 12: Best fitness and elapsed time are shown for the low, mid, and high values of the population size.

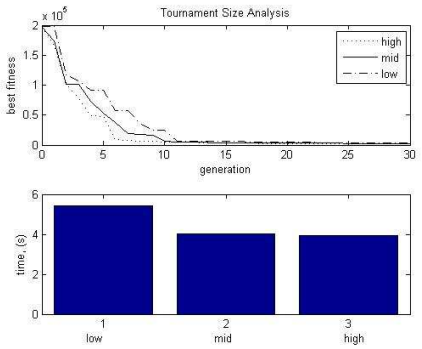


Figure 13: Best fitness and elapsed time are shown for the low, mid, and high values of the tournament size.

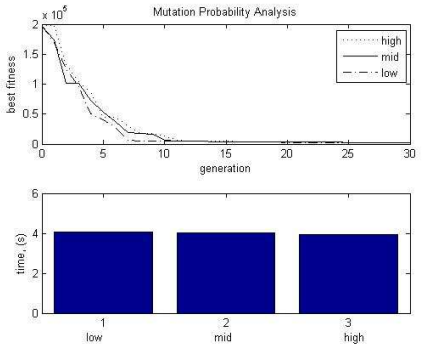


Figure 14: Best fitness and elapsed time are shown for the low, mid, and high values of the mutation probability.

minimum. Tournament size has little effect on computation time.

Mutation probability effected the best fitness of the population as seen in figure 14. High mutation causes the genetic algorithm to require more generations but found better paths. Low mutation found an acceptable path quickly, but didn't find the best path as quickly. More experimentation can be done in this area to determine what mutation probabilities provide the best paths with the least number of generations. Mutation probability has no effect on the computation time because mutation happens rarely and is easy to compute.

Figure 15 shows the effects of the variation on the mutation. The high and low max mutation distance seemed to have little effect on the genetic algorithm's success. Low mutation actually provided a best fitness faster, but it may be because high mutation wasn't necessary for this terrain model. The difference seen in the time to run 30 generations is due to the random nature of the genetic algorithm.

Figure 16 shows the effects of the probability of adding a waypoint. Add probability primarily effected the running time of the algorithm. This is because the paths were longer and more terrain needed to be checked for path collisions. The low and medium values had insignificant time differences. The mid range value is a safe value to use for most path planning applications.

Figure 17 shows the effect of the probability of deleting waypoints. Delete probability decreased the time required to run because the paths were shorter. Shorter paths required less computation to determine terrain intersections. All values seemed to converge at the

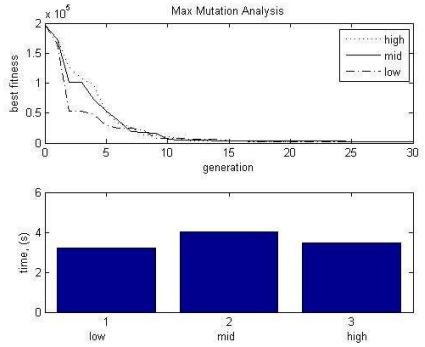


Figure 15: Best fitness and elapsed time are shown for the low, mid, and high values of the maximum mutation.

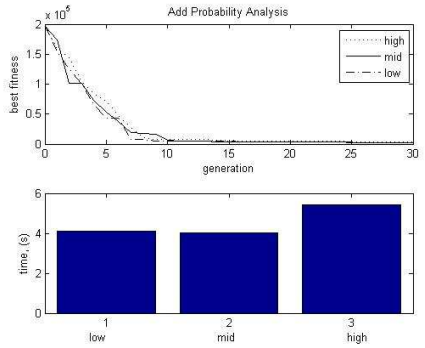


Figure 16: Best fitness and elapsed time are shown for the low, mid, and high values of the add probability.

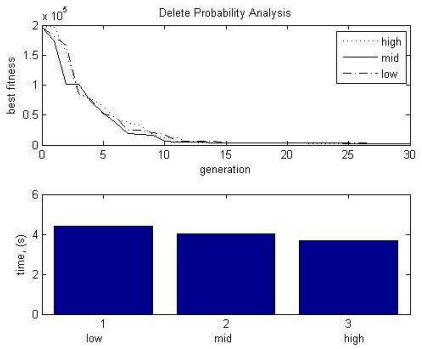


Figure 17: Best fitness and elapsed time are shown for the low, mid, and high values of the delete probability.

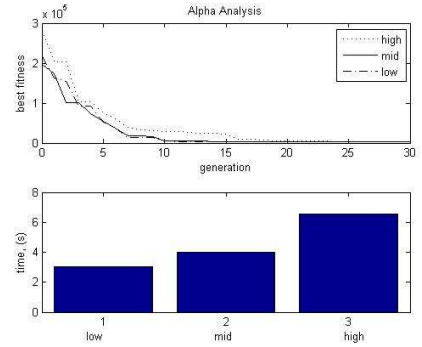


Figure 18: Best fitness and elapsed time are shown for the low, mid, and high values of alpha.

same rate. While increasing the delete probability helps speed the algorithm up, increasing to value too much will not allow for as much genetic variation because too much of the genetic information is deleted. The algorithm would potentially be more prone to finding local minima.

Figure 18 shows the effect of the α parameter. High values of α took longer to converge because the starting paths have large deviations from the straight line path. Alpha also increased the time required to run because the paths can be much longer requiring more terrain intersection computation. If possible, a very small value of alpha should be used because it decreased time to run the algorithm and provides the same quality of paths as higher values. If there is no good path between the start and end point within the bounding box created by these points then alpha can be increased. Additionally, if the start and end point are near the same coordinate in either the x or y dimension, the bounding box would be small. This should be remedied by changing the algorithm to have minimum region dimensions based on the distance between the start and end points.

Figure 19 shows the effect of the initial number of waypoints. The initial number of waypoints has the most impact of any of the parameters on the success of the genetic algorithm. More waypoints required more computation in all aspects of the genetic algorithm including crossover, fitness, mutation, and terrain collision detection therefore it increases the computation time significantly. However, using more waypoints helped the paths to be able to navigate more narrow corridors. More waypoints also helps the algorithm to search

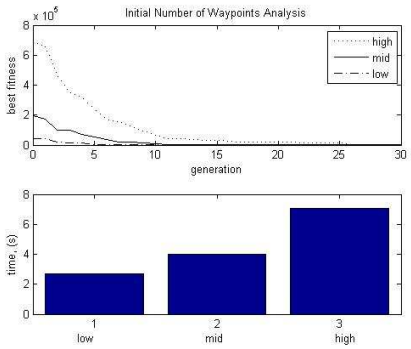


Figure 19: Best fitness and elapsed time are shown for the low, mid, and high values of the initial number of waypoints.

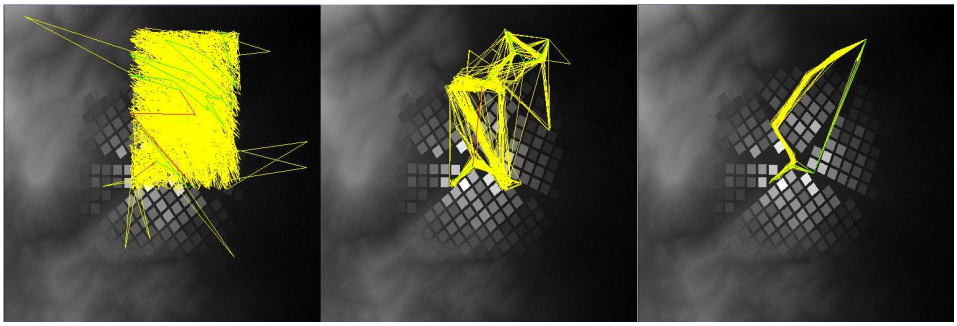


Figure 20: A sequence of screen shots from the genetic algorithm. Shots taken at 1st, 15th, and 30th Generations. Population paths are drawn in yellow. The best path drawn in green while terrain intersections are indicated by red.

more of the design space and provide a feasible path.

From the analysis given above it can be seen that the initial number of waypoints has the largest effect on the best fitness of the optimization. Tournament and population size also have a large effect on the on the success and run time of the algorithm.

2.1.2.4 Software Screenshots Figure 20 shows a sequence of screen shots from the generations of the genetic algorithm. The images are taken from the 1st, 15th and 30th generations. The yellow lines represent the population of paths while the green line is the path with the lowest fitness. Red lines are segments of the best path that intersect the terrain.

2.1.3 Reactive Path Planning

Despite having an effective a priori path planner, we cannot guarantee that the flight path will be free of obstacles. Our path planner assumes a perfect model of the terrain, but this assumption is not realistic. If an urban terrain model is missing a newly constructed building or a large antenna or tree, a path leading to a collision could result. Our canyon models are based on 10 m USGS data, which is fairly accurate, but which cannot represent small obstacles like trees and power lines. In addition, the GPS sensor used on the MAV has a constant bias that can be as large as 10 m. Path planners can produce a nominal

path prior to flight, but the MAV must also have the ability to sense and reactively avoid unanticipated obstacles and terrain in real time.

Reactive obstacle avoidance from a MAV platform is challenging because of the size and weight limitations for sensing and computation hardware imposed by the platform. The speed with which avoidance decisions must be made and carried out also causes difficulties. For obstacle avoidance in urban environments, we have developed a heuristic algorithm that utilizes a laser ranger to detect and avoid obstacles. The laser ranger points directly out the front of the MAV, and returns range data for objects directly in front of the MAV with a 3 Hz update. For our preliminary flight tests, we considered a simple scenario: a single unknown obstacle placed directly in the flight path.

2.1.3.1 Algorithm Consider the scenario shown in Figure 21 where obstacle avoidance is required. The MAV has a forward ground velocity V and a minimum turn radius R and is assumed to be tracking the given waypoint path at the time the obstacle is detected by the laser, which has a look ahead distance L. Figure 21 (a) shows the instant when the obstacle is detected by the laser ranger. The basic idea is to construct an internal map of obstacles detected by the laser and to modify the waypoint path to maneuver around the obstacles in the internal map. We will refer to the internal representation of obstacles as "map obstacles." When the laser detects the location of an obstacle, we are unsure about the size and height of the obstacle. We propose representing map obstacles as cylinders with radius R equal to the minimum turn radius of the MAV, and height equal to the current altitude of the MAV. As shown in Figure 21 (b), there are two alternate waypoint paths that maneuver around the map obstacle. The endpoints of the waypoint paths are selected so that the new waypoint paths are tangent to the obstacles in the internal map. As shown in Figure 22 (a), the new waypoints are located a distance $dR/\sqrt{d^2 - R^2}$ from the original waypoint path, where d is the turn away distance from the obstacle. If both waypoint waypoint paths are collision free, then the algorithm randomly selects between the two paths as shown in Figure 21 (c). Since the map obstacle may be smaller than the the actual obstacle, the laser may again detect the obstacle as it maneuvers on the modified path. If that is the case, a new map obstacle is added to the internal map as shown in Figure 21 (d). This process is repeated until the MAV maneuvers around the obstacle as shown in Figures 21 (e) and (f).

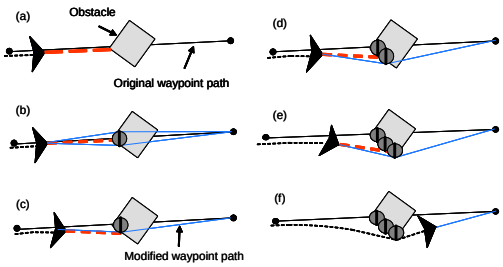


Figure 21: Obstacle avoidance algorithm. (a) The laser detects the obstacle. (b) A map obstacle of radius R is inserted into the map, and two candidate waypoint paths are constructed. (c) A modified waypoint path is randomly selected. (d) The obstacle is again detected by the laser and another map obstacle is constructed. (e-f) The process repeats until the MAV is able to maneuver around the obstacle.

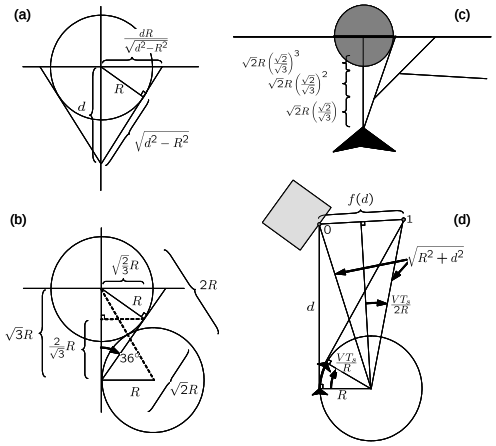


Figure 22: (a) The waypoint path is constructed so that it is perpendicular to the map obstacle. The radius R ensures collision free passage around the map obstacle. (b) The maximum heading change in waypoint paths is when the MAV must make a full bank to maneuver around the obstacle. (c) An approximation of the minimum distance required to avoid a straight wall if the laser is only sampled when the MAV is on the waypoint path. (d) the geometry used to calculate the distance between two consecutive laser updates.

If we assume zero wind, then the 2-D navigation for the MAV is given by

$$\begin{aligned}\dot{n} &= V\cos\chi \\ \dot{e} &= V\sin\chi \\ \dot{\chi} &= \frac{g}{V}\tan\phi,\end{aligned}$$

where g is the gravitational constant, and ϕ is the roll angle of the MAV. On most MAVs, the roll angle is limited between $-\bar{\phi} \leq \phi \leq \bar{\phi}$. We will assume that the roll dynamics of the MAV are sufficiently fast to assume near instantaneous transitions between $\pm\bar{\phi}$. Therefore, the minimum turn radius is given by $R = \frac{V^2}{g\tan\bar{\phi}}$.

We would like to establish a minimum turn away distance D so that we are guaranteed to avoid collision with a single rectangular obstacle. The first step is to determine the bounds on the forward and lateral motion of the MAV when it transitions from one waypoint path to the next.

Claim: After the insertion of a map obstacle, the MAV requires at most a forward distance of $\frac{2}{\sqrt{3}}R$ *and a lateral distance of* $\sqrt{\frac{2}{3}}R$ *to transition onto the new waypoint path while avoiding the map obstacle.*

Assuming the ability to roll instantaneously between $\pm\bar{\phi}$, the motion of the MAV during the transition can be constrained to lie on circles of radius R. As shown in [6], the path length of the transition increases monotonically with the angle between the old and new waypoint paths. Therefore, the forward and lateral distances are maximized when the angular separation is maximized, which occurs when instantaneous motion of the MAV follows a circle of radius R that just touches the map obstacle, as shown in Figure 22 (b). The claim follows directly from standard geometrical arguments. Note that the maximum angular separation is therefore given by $\theta = \tan^{-1}\frac{1}{\sqrt{2}} \approx 36°$.

Claim: Avoidance of a collision with a flat wall is guaranteed if the the turn away distance

D satisfies

$$D > \left(\frac{8+2\sqrt{6}}{2\sqrt{3}}\right) R. \tag{3}$$

Consider the worst-case scenario, shown in Figure 22 (c), of a MAV that is initially traveling perpendicular to a flat wall. The MAV detects an obstacle and inserts a waypoint at maximum angle $\tan^{-1}\frac{1}{\sqrt{2}}$. After aligning its heading with the waypoint path, the wall is again detected, a map obstacle is inserted, and a new waypoint with maximum angle $\tan^{-1}\frac{1}{\sqrt{2}}$ is planned. This scenario will repeat itself at most three times since $3\tan^{-1}\frac{1}{\sqrt{2}} > \frac{\pi}{2}$. Therefore, the maximum forward direction is bounded by

$$\sqrt{2}R\left(\left(\sqrt{\tfrac{2}{3}}\right)^1 + \left(\sqrt{\tfrac{2}{3}}\right)^2 + \left(\sqrt{\tfrac{2}{3}}\right)^3\right) = \left(\tfrac{8+2\sqrt{6}}{2\sqrt{3}}\right) R.$$

We note that the algorithm described above, requires that the laser detect points on the obstacle that are outside of the map obstacles as soon as they become visible. Is this feasible given the update rate of the laser? Let T_s be the time between laser updates.
Claim: The maximum distance between laser updates at a range of $d \leq L$ is given by

$$f(d) = 2\sqrt{R^2+d^2}\sin\left(\frac{VT_s}{2R}\right)$$

Assuming the vehicle is turning at its maximum rate, the change in heading between updates is $\frac{VT_s}{R}$. Utilizing the geometry depicted in Figure 22 (d), the calculation of $f(d)$ is straightforward. To ensure overlap of map obstacles between samples we require that $f(D) < R$ which implies that

$$T_s < \frac{2R}{V}\sin^{-1}\left(\frac{R}{2\sqrt{R^2+D^2}}\right).$$

For our airframes, typical values are $V = 13$ m/s, $R = 25$ m, which implies from (3) that $D = 93$ m and $T_s < 0.5$ s. The laser ranger sample period of 0.33 s satisfies this constraint, thus ensuring that map obstacles overlap between samples.

2.2 Objective 2: Robust Trajectory Generation

Unmanned aerial vehicles (UAVs), large and small, have demonstrated their usefulness in military applications. Furthermore, there are numerous potential uses for UAVs in civil and commercial applications and the prospects for broad impact are strong. To extend the usefulness of UAVs beyond their current applications, the capability to plan paths and to follow them accurately is of great importance. Unlike piloted vehicles, which rely on the pilot to navigate over demanding terrain or to avoid obstructions, UAVs rely on automation to provide this functionality. As applications such as urban surveillance and rural search and rescue require UAVs to fly down city streets surrounded by buildings or near the surface of abruptly changing mountainous terrain, the ability to follow pre-planned paths with precision is essential. For missions involving cooperation among a team of UAVs, precise path tracking is often crucial to achieving the cooperation objective.

For miniature aerial vehicles,[1] such as those of primary interest in this work, wind disturbances, dynamic characteristics, and the quality of sensing and control all limit the achievable tracking precision. For MAVs wind speeds are commonly 20 to 60 percent of the desired airspeed. Effective path tracking strategies must overcome the effect of this ever present disturbance. For most fixed-wing MAVs, the minimum turn radius is in the range of 10 to 50 m. This places a fundamental limit on the spatial frequency of paths that can be tracked. Thus, it is important that the path tracking algorithms utilize the full capability of the MAV. Finally, high-resolution state sensors with high-frequency updates are not typically available for MAVs. Successful tracking approaches must exploit fully those sensors that are readily available.

Several approaches have been proposed for UAV trajectory tracking. An approach for tight tracking of curved trajectories is presented in [9]. For straight-line paths, the approach approximates PD control. For curved paths, an additional anticipatory control element that improves the tracking capability is implemented. The approach accommodates the addition of an adaptive element to account for disturbances such as wind. This approach is validated with flight experiments.

In [10], Kaminer et al. describe an integrated approach for developing guidance and control algorithms for autonomous vehicle trajectory tracking. Their approach builds upon the theory of gain scheduling and produces controllers for tracking trajectories that are defined in an inertial reference frame. The approach is illustrated through simulations of a small UAV.

Implicit in the notion of trajectory tracking is that the vehicle is commanded to be in a particular location at a particular time and that this location typically varies in time, thus causing the vehicle to move in the desired fashion. With fixed-wing MAVs, the desired position is constantly moving (at the desired air speed). The approach of tracking a moving point can result in significant problems for MAVs if disturbances, such as those due to wind, are not accounted for properly. If the MAV is flying into a strong wind (relative to its commanded ground speed), the progression of the trajectory point must be slowed accordingly. Similarly, if the MAV is flying down wind, the speed of the tracking point must be increased to keep the MAV from overrunning the desired position. Given that wind disturbances vary and are often not easily predicted, trajectory tracking can be very challenging in anything other than calm conditions.

Rather than pursuing the trajectory tracking approach, this report explores path following where the objective is to be *on the path* rather than at a certain point at a particular time. With path following, the time dependence of the problem is removed. In [11, 12],

[1] We consider miniature aerial vehicles to be those with wingspans in the 0.3 m to 2 m range and micro aerial vehicles to be those with wingspans under 0.3 m. Here the abbreviation MAV denotes *miniature* aerial vehicle.

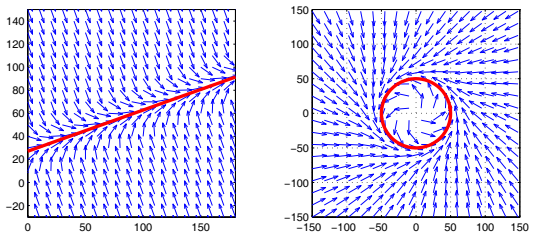


Figure 23: This figure illustrates the vector field idea for straight line and circular path following. The the desired course of the MAV is specified the by direction of the vector field. For straight line paths, the vector field is relatively constant far from the line, and transitions onto the line as the relative distance becomes closer. A similar idea is used for circular orbits.

performance limits for reference-tracking and path-following controllers are investigated and the difference between them is highlighted. It is shown that there is not a fundamental performance limitation for path following for systems with unstable zero dynamics as there is for reference tracking.

Building on the work presented in [13] on maneuver modified trajectory tracking, Encarnação and Pascoal develop an approach that combines the features of trajectory tracking and path following for marine vehicles [14]. Similar to this work is that of Skjetne, et al. [15] which develops an output maneuvering method composed of two tasks: forcing the output to converge to the desired path and then satisfying a desired speed assignment along the path. The method is demonstrated using a marine vessel simulation. Ref. [16] presents a path following method for UAVs that provides a constant line of sight between the UAV and an observation target.

The work presented in this report builds on the concept of path following through the construction of vector fields surrounding the path to be followed. The vectors of the fields provide course commands to guide the MAV toward the desired path. As with other path following methods, the objective is not to track a moving point, but to get onto the path while flying at a prescribed airspeed. A unique contribution of this work is the utilization of course measurements in the path following control, which in combination with the vector field strategy, guarantees that tracking errors asymptotically approach zero even in the presence of constant wind disturbances. Implementation of the approach is feasible on small MAVs and experimental results validate the potential value of the approach for MAVs flying in windy conditions.

2.2.1 Problem Description

The objective of this work is to develop a method for accurate path following for MAVs in the presence of wind. The method calculates a vector field around the path to be tracked. The vectors in the field are directed toward the path to be followed and represent the desired direction of flight. The vectors in the field serve as course commands to the MAV. The method is currently applicable to paths composed of straight lines and arcs. This restriction is insignificant for most practical applications. Figure 23 shows examples of vector fields for linear and circular paths.

The notion of vector fields is similar to that of potential fields, which have been widely used as a tool for path planning in the robotics community (see e.g., [17]). It has also been suggested in [18] that potential fields can be used in UAV navigation for obstacle and collision avoidance applications. The method of [18] provides a way for groups of UAVs to

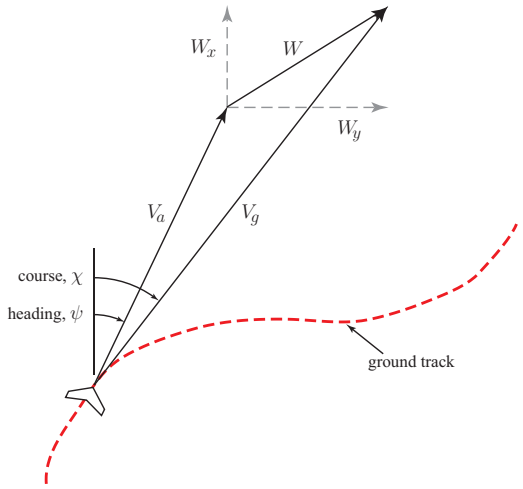


Figure 24: This figure shows the relationship between the airspeed V_a, the windspeed W, and ground speed V_g, as well as the relationship between heading ψ and course χ.

use the gradient of a potential field to navigate through heavily populated areas safely while still aggressively approaching their targets. Vector fields are different from potential fields in that they do not necessarily represent the gradient of a potential. Rather, the vector field simply indicates a desired direction of travel.

This report considers the navigation of a fixed-wing MAV with the assumption that altitude and airspeed (V_a) are held constant (or nearly so) by the control of the longitudinal dynamics. The following is a simple model of the navigational dynamics that will be used to study the path following behavior of the proposed approach:

$$\dot{x} = V_a \cos\psi + W_x \tag{4}$$
$$\dot{y} = V_a \sin\psi + W_y \tag{5}$$

where (W_x, W_y) represent the x and y components of the wind velocity. Heading (ψ) will be controlled by the vector field path following approaches presented in this report. An alternative representation of these equations can be developed by noting the relationship between groundspeed (V_g), airspeed (V_a), and wind speed (W) as depicted in Figure 24:

$$\dot{x} = V_{ax} + W_x = V_{gx} \tag{6}$$
$$\dot{y} = V_{ay} + W_y = V_{gy}. \tag{7}$$

Drawing on Equations (6) and (7) and the definition of course (χ) shown in Figure 24, Equations (4) and (5) can be expressed as

$$\dot{x} = V_g \cos\chi \tag{8}$$
$$\dot{y} = V_g \sin\chi \tag{9}$$

The key distinction is that the equations of motion are expressed in terms of groundspeed and course and are independent of the wind velocity. When ground-based measurements are used in conjunction with the vector field approach to control the path of the vehicle, wind-disturbance rejection is improved dramatically, which is vitally important for small, low-speed MAVs. We will assume that the MAV is equipped with an autopilot that implements

a course-hold loop and that the resulting dynamics are represented by the first-order system

$$\dot{\chi} = \alpha\left(\chi^c - \chi\right), \tag{10}$$

where χ^c is the commanded course, and α is a known positive constant.

In the development and analysis of the vector field approach that follows, two primitive path types are considered: straight lines and circular orbits. Circular arcs are treated similarly to complete orbits. The approach is easily extended to paths composed of multiple segments of arcs, orbits, and straight lines.

2.2.2 Technical Approach

2.2.2.1 Straight Path Following Consider the straight-line path shown in Figure 25. To follow this path, a desired-course vector field is constructed. Let y be the lateral distance of the MAV from the path, and let χ be the difference between the direction of the path and the course of the MAV. Our objective is to construct the vector field so that when y is large the MAV is directed to approach the path with course angle χ^∞, and that as y approaches zero, the course χ also approaches zero. Toward that end, define the desired course of the MAV as

$$\chi^d(y) = -\chi^\infty \frac{2}{\pi}\tan^{-1}(ky), \tag{11}$$

where k is a positive constant that influences the rate of the transition from χ^∞ to zero. If χ^∞ is restricted to be in the range $\chi^\infty \in (0, \frac{\pi}{2}]$ then clearly

$$-\frac{\pi}{2} < \chi^\infty \frac{2}{\pi}\tan^{-1}(ky) < \frac{\pi}{2}$$

for all values of y. Therefore since $\tan^{-1}(\cdot)$ is an odd function and $\sin(\cdot)$ is odd over $(-\frac{\pi}{2}, \frac{\pi}{2})$ we can use the Lyapunov function $W_1(y) = \frac{1}{2}y^2$ to argue that if $\chi = \chi^d(y)$, then $y \to 0$ asymptotically. Evaluating the Lie derivative of W_1 under the assumption that $\chi = \chi^d(y)$ gives

$$\begin{aligned}\dot{W}_1 &= V_g y \sin(\chi^d(y)) \\ &= -V_g y \sin\left(\chi^\infty \frac{2}{\pi}\tan^{-1}(ky)\right),\end{aligned}$$

which is less than zero for $y \neq 0$.

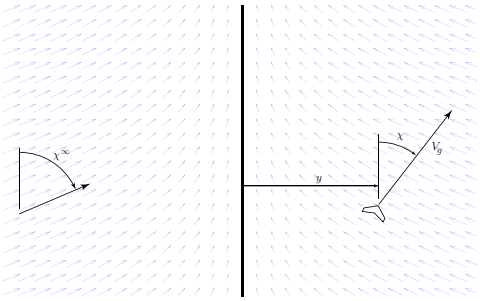


Figure 25: Vector field for straight line path following. Far away from the waypoint path, the vector field is directed with an angle χ^∞ from the perpendicular to the path.

In this work we use a sliding mode approach to render the set

$$S = \{(y, \chi) : \chi = \chi^d(y)\}$$

positively invariant and to ensure that the system trajectory reaches S in finite time. Let $\tilde{\chi} \triangleq \chi - \chi^d(y)$ and differentiate to obtain

$$\begin{aligned}\dot{\tilde{\chi}} &= \dot{\chi} - \dot{\chi}^d(y) \\ &= \alpha(\chi^c - \chi) + \chi^\infty \frac{2}{\pi} \frac{k}{1+(ky)^2} V_g \sin\chi. \end{aligned} \tag{12}$$

Let $W_2 = \frac{1}{2}\tilde{\chi}^2$ and take the derivative to obtain

$$\begin{aligned}\dot{W}_2 &= \tilde{\chi}\dot{\tilde{\chi}} \\ &= \tilde{\chi}\left(\dot{\chi} - \dot{\chi}^d(y)\right) \\ &= \tilde{\chi}\left(\alpha(\chi^c - \chi) + \chi^\infty \frac{2}{\pi} \frac{k}{1+(ky)^2} V_g \sin\chi\right). \end{aligned}$$

If we choose the control signal as

$$\chi^c = \chi - \frac{1}{\alpha}\chi^\infty \frac{2}{\pi} \frac{k}{1+(ky)^2} V_g \sin\chi - \frac{\kappa}{\alpha}\text{sign}(\tilde{\chi}), \tag{13}$$

where

$$\text{sign}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x = 0 \\ -1 & \text{if } x < 0 \end{cases}$$

and $\kappa > 0$, then

$$\dot{W}_2 \leq -\kappa\,|\tilde{\chi}| \tag{14}$$

from which we conclude that $\tilde{\chi} \to 0$ in finite time [19].

It is well known that the sign function leads to chattering in the control [19]. To mitigate the adverse effects of chattering, the control signal (13) is replaced with

$$\chi^c = \chi - \frac{1}{\alpha}\chi^\infty \frac{2}{\pi} \frac{k}{1+(ky)^2} V_g \sin\chi - \frac{\kappa}{\alpha}\text{sat}\left(\frac{\tilde{\chi}}{\epsilon}\right), \tag{15}$$

where

$$\text{sat}(x) = \begin{cases} x & \text{if } |x| \leq 1 \\ \text{sign}(x) & \text{otherwise} \end{cases},$$

and $\epsilon > 0$ defines the width of the boundary region around the sliding surface.

Theorem 2.2 *The system of equations given by (9) and (12), where χ^c is given by (15) and $\chi^d(y)$ is given by (11) is globally exponentially stable if $0 < \chi^\infty < \frac{\pi}{2}$ and*

$$\frac{\kappa}{\epsilon V_g}\left(\frac{4\chi^\infty k}{\pi}\right) > 1. \tag{16}$$

The stability condition (16) makes sense physically. If V_g is large, than the turning radius of the MAV increases making it more difficult to remain within boundary layer. Similarly small χ^∞ or small k implies that there will be a rapid change in χ^d close to the waypoint path which will be difficult to track. Small ϵ makes the feedback control high gain which enhances tracking. It is interesting to note that (16) is independent of α. However, from (15) it is clear that to avoid unrealistic control effort, κ is required to be proportional to α. Therefore, (16) is easier to satisfy if α is large, implying that the on-board autopilot can quickly track course changes.

Proof:

If $|\tilde{\chi}| \geq \epsilon$, then χ^c is equivalent to (13) which results in (14) implying that the set

$$S_\epsilon \triangleq \{|\tilde{\chi}| \leq \epsilon\} \tag{17}$$

is positively invariant and that $\tilde{\chi}$ converges to S in finite time.

It remains to show that inside S_ϵ the system trajectories converge to the origin $(y, \tilde{\chi}) = (0, 0)$. Toward that end define the Lypunov function candidate

$$W = \frac{1}{2}y^2 + \frac{1}{2}\tilde{\chi}^2.$$

Differentiating we obtain

$$\begin{aligned}
\dot{W} &= y\dot{y} + \tilde{\chi}\dot{\tilde{\chi}} \\
&= yV_g \sin\left(\chi^d(y) + \tilde{\chi}\right) \\
&\quad + \tilde{\chi}\alpha\left(\chi^c - \chi + \frac{1}{\alpha}\frac{2\chi^\infty}{\pi}\frac{k}{1+(ky)^2}V_g \sin\chi\right).
\end{aligned}$$

Inside the boundary region we have

$$\chi^c = \chi - \frac{1}{\alpha}\chi^\infty\frac{2}{\pi}\frac{k}{1+(ky)^2}V_g\sin\chi - \frac{\kappa}{\alpha}\frac{\tilde{\chi}}{\epsilon}.$$

Therefore

$$\begin{aligned}
\dot{W} &= V_g y \sin\left(\chi^d(y) + \tilde{\chi}\right) - \frac{\kappa}{\epsilon}\tilde{\chi}^2 \\
&= -\frac{\kappa}{\epsilon}\tilde{\chi}^2 + V_g y \sin\left(\chi^d(y)\right) \\
&\qquad + V_g y\left(\sin\left(\chi^d(y) + \tilde{\chi}\right) - \sin\left(\chi^d(y)\right)\right) \\
&\leq -\frac{\kappa}{\epsilon}\tilde{\chi}^2 + V_g y \sin\left(\chi^d(y)\right) \\
&\qquad + V_g |y| \left|\sin\left(\chi^d(y) + \tilde{\chi}\right) - \sin\left(\chi^d(y)\right)\right|.
\end{aligned}$$

Noting that

$$\begin{aligned}
&\left|\sin\left(\chi^d(y) + \tilde{\chi}\right) - \sin\left(\chi^d(y)\right)\right| \\
&\quad = \left|\sin\chi^d(y)\cos\tilde{\chi} + \cos\chi^d(y)\sin\tilde{\chi} - \sin\chi^d(y)\right| \\
&\quad = \left|\sin\chi^d(y)(\cos\tilde{\chi} - 1) + \cos\chi^d(y)\sin\tilde{\chi}\right| \\
&\quad \leq |\cos\tilde{\chi} - 1| + |\sin\tilde{\chi}| \\
&\quad \leq 2|\tilde{\chi}|,
\end{aligned}$$

we get

$$\begin{aligned}
\dot{W} &\leq -\frac{\kappa}{\epsilon}\tilde{\chi}^2 + 2V_g|y||\tilde{\chi}| + V_g y\sin\left(\chi^d(y)\right) \\
&= -\frac{\kappa}{\epsilon}\tilde{\chi}^2 + 2V_g|y||\tilde{\chi}| - V_g y\sin\left(\frac{2\chi^\infty}{\pi}\tan^{-1}(ky)\right).
\end{aligned}$$

Let

$$\phi(y) \triangleq y\sin\left(\frac{2\chi^\infty}{\pi}\tan^{-1}(ky)\right),$$

and note that

$$\begin{aligned}\left|\frac{\partial\phi}{\partial y}\right| &= \left|\sin\left(\frac{2\chi^\infty}{\pi}\tan^{-1}(ky)\right)\right. \\ &\qquad \left. + y\cos\left(\frac{2\chi^\infty}{\pi}\tan^{-1}(ky)\right)\frac{2\chi^\infty}{\pi}\frac{k}{1+(ky)^2}\right| \\ &\leq \left|\frac{2\chi^\infty}{\pi}\tan^{-1}(ky)\right| + \left|\frac{2\chi^\infty}{\pi}\frac{ky}{1+(ky)^2}\right| \\ &\leq 2\,|y|\,\frac{2\chi^\infty}{\pi}k.\end{aligned}$$

Using the fact that if f and g are continuous functions such that $f(0) = g(0)$ and $|f'(x)| \leq |g'(x)|$ then $|f(x)| \leq |g(x)|$ we get

$$\begin{aligned}\dot{W} &\leq -\frac{\kappa}{\epsilon}\tilde{\chi}^2 + 2V_g\,|y|\,|\tilde{\chi}| - 2V_g\frac{2\chi^\infty}{\pi}ky^2 \\ &= -V_g\begin{pmatrix}|\tilde{\chi}| & |y|\end{pmatrix}\begin{pmatrix}\frac{\kappa}{\epsilon V_g} & -1 \\ -1 & \frac{4\chi^\infty}{\pi}k\end{pmatrix}\begin{pmatrix}|\tilde{\chi}| \\ |y|\end{pmatrix} \qquad (18)\end{aligned}$$

which if negative definite if Equation (16) is satisfied. Exponential stability comes from the fact that W and the right hand side of (18) are quadratic [19]. ■

2.2.2.2 Orbit Following The algorithm for circular orbits creates vector fields in a manner similar to the straight-line algorithm. Consider the desired orbit path shown in Figure 26. In this discussion, a counter-clockwise orbit will be considered. The development for clockwise orbits is similar with the exception of several sign changes. The desired orbit is assumed to have a known center and radius r. When the distance between the MAV and the center of the orbit is large, it is desirable for the MAV to fly toward the orbit center. If we define d as the radial distance of the MAV from the center of the orbit, then when d is significantly larger than r the desired course is

$$\chi^d \approx \gamma - \pi \qquad (19)$$

where γ is defined as the angular position of the MAV with respect to the orbit center as shown in Figure 26.

When $d = r$ the desired course is $\chi^d = \gamma - \frac{\pi}{2}$. Therefore let the desired course be given by

$$\chi^d(d) = \gamma - \frac{\pi}{2} - \tan^{-1}\left(k\frac{d-r}{r}\right), \qquad (20)$$

where $k > 0$ is a constant that specifies the rate of transition from $\gamma - \pi$ to $\gamma - \frac{\pi}{2}$.

For orbit following, it is convenient to change the navigational dynamics to polar coordinates in terms of d and γ where the center of the orbit is the origin. From Figure 26, $x = c_x + d\cos\gamma$ and $y = c_y + d\sin\gamma$, where (c_x, c_y) is the center of the orbit. Taking the derivative and substituting into Equations (8) and (9) gives

$$\dot{d} = V_g\cos(\chi - \gamma) \qquad (21)$$

$$\dot{\gamma} = \frac{V_g}{d}\sin(\chi - \gamma) \qquad (22)$$

where the V_g and χ are the ground track speed and relative course, respectively. We will again assume that the course dynamics are given by

$$\dot{\chi} = \alpha(\chi^c - \chi). \qquad (23)$$

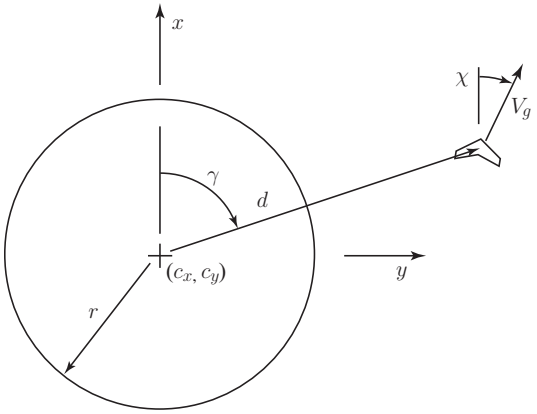


Figure 26: Vector field geometry for orbit tracking. When the radial distance to the MAV is much greater than the orbit radius the desired course is calculated so that the MAV is directed toward the orbit center. As the radial distance becomes smaller, the desired course becomes tangential to the orbit

Defining $\tilde{d} = d - r$ then we can argue that $\tilde{d} \to 0$ asymptotically when $\chi = \chi^d(d)$ by using the Lyapunov function $W_1 = \frac{1}{2}\tilde{d}^2$, whose Lie derivative is

$$\begin{aligned}\dot{W}_1 &= V_g \tilde{d} \cos\left(-\frac{\pi}{2} - \tan^{-1}\left((k/r)\tilde{d}\right)\right) \\ &= -V_g \tilde{d} \sin\left(\tan^{-1}\left((k/r)\tilde{d}\right)\right).\end{aligned}$$

$\dot{W}_1$ is less than zero for $\tilde{d} \neq 0$ since the argument of sin is in the set $(-\pi/2, \pi/2)$ for all $\tilde{d}$, implying that $\tilde{d} \to 0$ asymptotically.

We will again use a sliding mode approach to render the set

$$S = \{(\tilde{d}, \chi) : \chi = \chi^d(d)\}$$

positively invariant and to ensure that the system trajectory reaches S in finite time. As before, define $\tilde{\chi} = \chi - \chi^d(d)$ and differentiate to obtain

$$\begin{aligned}\dot{\tilde{\chi}} &= \alpha(\chi^c - \chi) - \dot{\chi}^d(d) \\ &= \alpha(\chi^c - \chi) - \frac{V_g}{d}\sin(\chi - \gamma) + \beta V_g \cos(\chi - \gamma),\end{aligned}$$

where

$$\beta = \frac{k/r}{1 + \left((k/r)\tilde{d}\right)^2}$$

has been defined for brevity. Letting $W_2 = \frac{1}{2}\tilde{\chi}^2$ gives

$$\dot{W}_2 = \tilde{\chi}\left(\alpha(\chi^c - \chi) - \frac{V_g}{d}\sin(\chi - \gamma) + \beta V_g \cos(\chi - \gamma)\right).$$

If we choose the control signal as

$$\chi^c = \chi + \frac{V_g}{\alpha d}\sin(\chi - \gamma) - \frac{\beta}{\alpha}V_g \cos(\chi - \gamma) - \frac{\kappa}{\alpha}\text{sign}(\tilde{\chi}), \tag{24}$$

where $\kappa > 0$, then

$$\dot{W}_2 \leq -\kappa \left|\tilde{\chi}\right| \tag{25}$$

from which we conclude that $\tilde{\chi} \to 0$ in finite time [19].

To avoid chattering in the orbit case we replace (24) with

$$\chi^c = \chi + \frac{V_g}{\alpha d}\sin(\chi - \gamma) - \frac{\beta}{\alpha}V_g\cos(\chi - \gamma) - \frac{\kappa}{\alpha}\text{sat}\left(\frac{\tilde{\chi}}{\epsilon}\right), \tag{26}$$

where $\epsilon > 0$ defines the width of the boundary region around the sliding mode.

Theorem 2.3 *The system of equations given by (21) and (23), where χ^c is given by (26) and $\chi^d(d)$ is given by (20) is globally exponentially stable if*

$$\frac{2\kappa k}{\epsilon V_g r} > 1. \tag{27}$$

Proof:

The proof follows a similar line of reasoning as the proof of Theorem 2.2. If $|\tilde{\chi}| \geq \epsilon$, then the set $S_\epsilon = \{|\tilde{\chi}| \leq \epsilon\}$ is positively invariant and $\tilde{\chi}$ converges to S_ϵ in finite time. Inside the boundary region we use the Lyapunov function candidate

$$W = \frac{1}{2}\tilde{d}^2 + \frac{1}{2}\tilde{\chi}^2$$

to obtain

$$\dot{W} = -\frac{\kappa}{\epsilon}\tilde{\chi}^2 + V_g\tilde{d}\sin\left(\hat{\chi}^d(d) + \tilde{\chi}\right)$$

where $\hat{\chi}^d(d) \triangleq -\tan^{-1}\left((k/r)\tilde{d}\right)$. Following steps that are similar to the proof of Theorem 2.2 we get

$$\begin{aligned}
\dot{W} &\leq -\frac{\kappa}{\epsilon}\tilde{\chi}^2 + 2V_g\left|\tilde{d}\right|\left|\tilde{\chi}\right| + V_g\tilde{d}\sin\left(\hat{\chi}^d(d)\right) \\
&= -\frac{\kappa}{\epsilon}\tilde{\chi}^2 + 2V_g\left|\tilde{d}\right|\left|\tilde{\chi}\right| - V_g\tilde{d}\sin\left(\tan^{-1}\left((k/r)\tilde{d}\right)\right) \\
&\leq -\frac{\kappa}{\epsilon}\tilde{\chi}^2 + 2V_g\left|\tilde{d}\right|\left|\tilde{\chi}\right| - 2V_g\frac{2k}{r}\tilde{d}^2 \\
&= -V_g\begin{pmatrix}\left|\tilde{\chi}\right| & \left|\tilde{d}\right|\end{pmatrix}\begin{pmatrix}\frac{\kappa}{\epsilon V_g} & -1 \\ -1 & \frac{2k}{r}\end{pmatrix}\begin{pmatrix}\left|\tilde{\chi}\right| \\ \left|\tilde{d}\right|\end{pmatrix}
\end{aligned}$$

which if negative definite if (27) is satisfied. ■

2.2.2.3 Combining Straight Lines and Orbits Many paths planned for MAV flight can be approximated by combinations of straight-line segments and circular arcs [7]. Figure 27 shows how combined paths can be utilized with waypoint planning to fly paths that preserve equal path lengths, fly directly over the waypoints, or turn in order to minimize flight time. There are also a number of other situations where a combination would be desirable. For example, following a perimeter with irregular geometry could be done effectively by approximating its geometry with a series of lines and arcs.

When combining straight and arc segments, an approach for constructing the vector field must be developed. In order to avoid the possibility of multiple sinks, dead zones, and singularities that are inherent in the combination of vector fields, only the vector field for

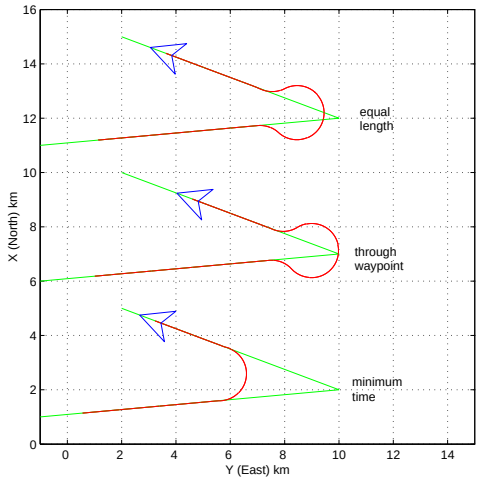


Figure 27: Straight line paths and circular orbits can be combined to produce a variety of paths. The top figure shows the combination of two straight line paths and three circular orbits that are arranged so that if the MAV is on the path, the path length is equal to the original waypoint path. The middle figure arranges the orbits so that the desired path transitions over the waypoint. In the bottom figure a single orbit is used to transition between straight line segments.

the current segment or arc to be followed is calculated. For a multi-segmented path, the vector field changes each time the MAV reaches the end of a segment or arc. Once the MAV has reached the end of a segment or arc, the entire vector field changes to direct the MAV onto the next segment or arc. No two fields are combined, thus eliminating any issues related to the combining of fields.

The method for determining when to change the vector field must be specified. There are a number of methods for doing this. One way is to detect when the MAV is within a predetermined distance from the end of the segment or arc. This works well for transitioning out of a straight path segment. For transitioning out of an arc, monitoring the angular travel of the MAV has proven successful. Using this approach, the MAV transitions to the next path segment when the angle through which the MAV has flown is equal to the included angle of the arc.

2.2.3 Wind estimation

Following the designated path in the presence of environmental disturbances is a key to mission success. A wind estimation and compensation algorithms were developed assuming that wind is an additive position error as shown in figure 28.

The wind estimation algorithm was designed to be used in conjunction with an adaptive trajectory tracker developed during Phase II based on the concept of "reachability regions." At some time T in the future, known as the "lookahead time," the UAV has a fixed set of points it can have reached. This set of points is known as the reachability region. Illustrated in figure 29 are lookahead regions for various values of T.

To track a path, the point in the reachability region that is closest to the desired path is determined, and the UAV control surfaces are commanded such that the vehicle will reach that point at time T. To compensate for wind, a reachability region that accounts for the effects of wind is estimated. At time T, the wind will have blown the UAV off of it course by the wind vector scaled by the lookahead time, assuming the wind vector is measured in

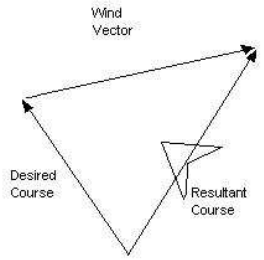


Figure 28: The effect of wind on the course of a UAV

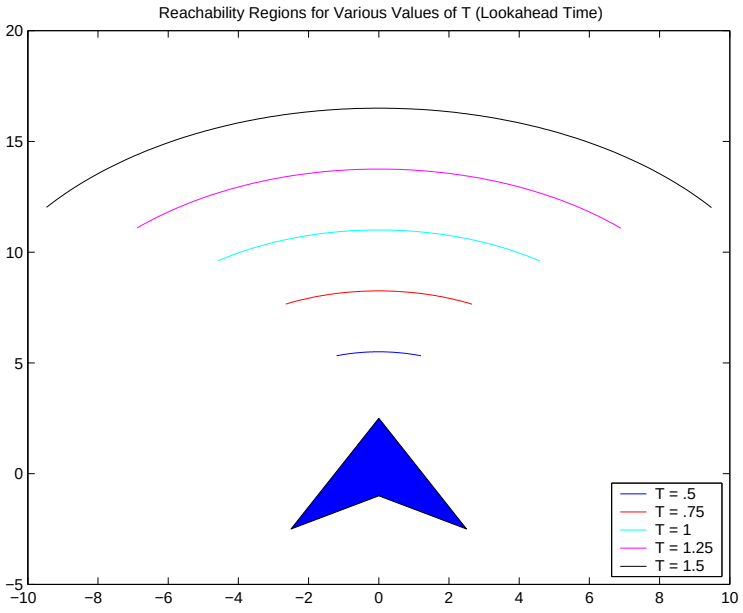


Figure 29: Reachability regions at various times, no wind

units/second. Therefore, the new reachability region is the no-wind region translated by this scaled wind vector.

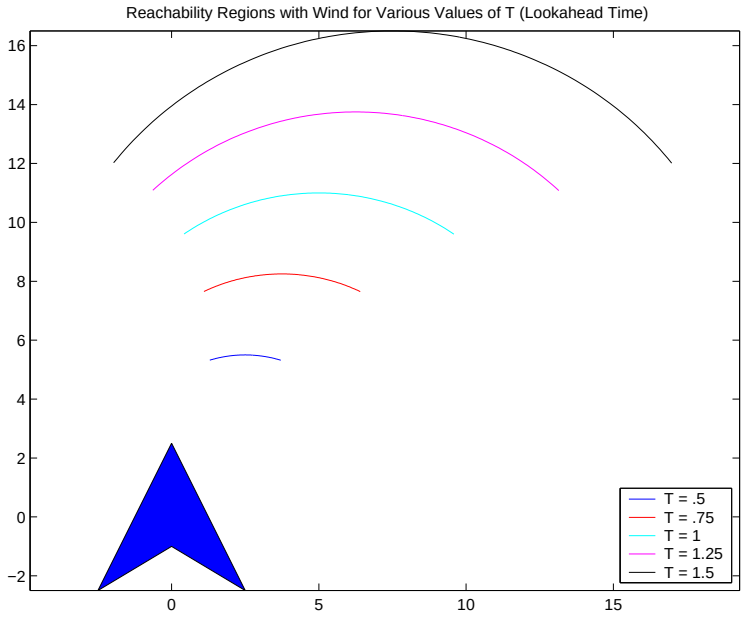


Figure 30: Reachability regions at various times, offset by wind

This approach requires that we estimate the wind vector using sensors already onboard the UAV. One approach investigated was to compare that the difference between airspeed and ground speed for various values of ψ. The wind direction is directly related to ψ. When ψ and the wind direction are the same (or differ by 180 degrees), the difference between ground speed and airspeed will be at a minimum (or a maximum). If the wind is constant, the absolute values of the maximum and minimum readings should be equal. In figure 31, samples are taken from the simulation of a UAV in an orbit routine, with constant winds of various magnitudes. When the UAV is flying directly upwind in this example, ψ measures $-\pi/2$. This graph peaks at $\pi/2$, however, because our goal is to measure the direction of wind travel rather than the direction of its origin. Samples are taken once a second over a 250-second simulation and averaged. The motivation for averaging and storing samples in evenly-spaced phase bins originates from the need to limit autopilot memory and computing power used to estimate wind. It also has the desired effect of reducing wind noise. In order to average a sample into a bin, we need two arrays. The first array holds the averaged sample values. The second array holds the number of samples stored in each bin.

2.2.3.1 Algorithm In order to extract wind direction and speed information from the bins (phase and magnitude of the sinusoid, respectively), we employ a FFT variant, that extracts the necessary measurements while being computationally inexpensive. Defining n as the width (in degrees) of each bin, the algorithm is:

$$Cosines = \sum_{k=1}^{360/n} \cos(360 * k/n) * values[k]$$

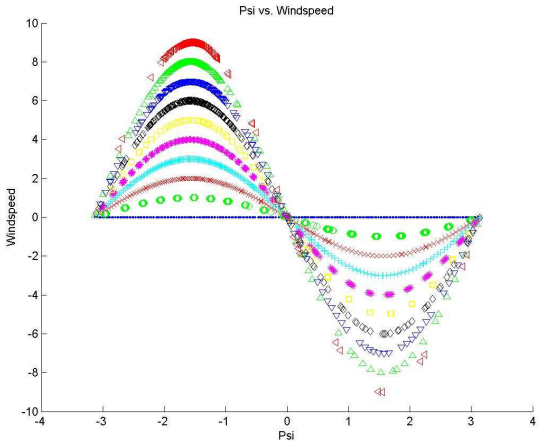


Figure 31: Ψ vs. windspeed

$$Sines = \sum_{k=1}^{360/n} \sin(360 * k/n) * values[k]$$

$$Wind\_direction = \arctan(Sines/Cosines)$$

$$Wind\_speed = (n/2) * \sqrt{Cosines^2 + Sines^2}$$

Incomplete data may be an issue when the UAV is flying downwind and fewer samples are possible per value of ψ. It is also likely that the UAV may not be commanded to orbit or follow a path in which wind samples can be taken for all values of ψ. Therefore we need a method to give us a good wind estimate even with incomplete data. In ideal situations, the data should repeat itself every 2π, and that the value in bin B should be negative of the value found in the bin 180 degrees apart from it. Therefore, if a bucket does not contain data but its "sister bucket" (180 degrees away) does contain data, we multiply the sister data value by -1 to estimate it. This increases the rise time of the wind estimation (although wind estimation is related more closely to values of heading at which we have taken samples than time: 180 degrees of heading variation gives a decent estimate, and 360 degrees of heading variation gives a better one).

The simplest solution to this problem is to monitor the maximum (or negative of the minimum) wind speed reading. Once the UAV has faced directly into or directly away from the wind, this reading should be the wind speed. The value of ψ at which this reading is taken is the wind direction (or wind direction + 180 degrees). This method works very well for determining the wind magnitude in the ideal situation: no noise, no wind gusts, and no other abnormalities. The rise time to correct values is fast (as soon as the UAV faces into/away from direct wind). Even in the ideal situation, however, accuracy of the wind direction is limited by the bin spacing. If bins are spaced every 20 degrees, the direction will be accurate to within 10 degrees.

In non-ideal situations, a large gust causes this method may result in inaccurate results samples are averaged. The rapidity of this depends on the UAV course. Beyond averaging, data outside of the largest-valued bin has no impact on our wind estimate. The method could be improved by averaging the maximum and minimum if the absolute values of those

readings are close, but the FFT algorithm shows more promise of combining computational simplicity with accuracy.

2.2.3.2 GPS Heading vs. Psi There are two way to estimate the heading of a UAV. The first method, that was implemented on the Kestral autopilot, is derived from GPS readings, sampled at the GPS sampling rate (currently 1 Hz). This gives the direction that the UAV is moving across the ground. However, this direction is not always the same as the direction where the nose of the airplane is pointed. Because GPS data is sampled only once every second, error is introduced in the readings (the UAV could be traveling more slowly at the beginning of the second than at the end, but the equations model that it traveled the whole second at the same speed). This could be improved by using a GPS sensor that provides data at a higher frequency.

2.2.3.3 Comparisons of Algorithm Performance After testing the wind estimation algorithm on ideal data, we incorporated additional elements into the wind simulation that make it more realistic. These include sensor noise, wind noise, and discrete sensor sampling times. The simulation includes the option to add noise to all sensors, and to change their sample times. Although the graphs are plotted versus time, the more important thing to notice is the relationship between data plotted and the range of values that ψ or GPS heading has covered.

If the heading has ranged across 180 degrees of readings, a measurement has been taken either directly into or directly away from the wind, which is important in a good wind estimate. If the heading has ranged across 360 degrees or more, the wind estimate is generally even better. Of the methods considered, the best combination is wind estimation using the FFT algorithm, -180-degree averaging, GPS heading, and the last sample's GPS velocity. All data was collected while commanding the UAV with the orbit algorithm.

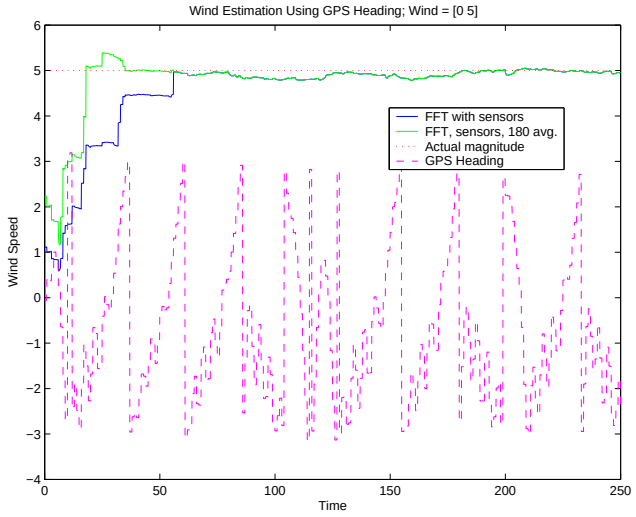


Figure 32: Wind Speed Estimation Using GPS Heading, FFT, Sensors, Comparing -180 Averaging to Measurements Only

2.3 Objective 4: Collision Avoidance Sensors and Algorithms

The original objective in this project was to flight test the path planning algorithms using simulated sensors and a simulated environment. This objective was accomplished during the first year of the project. During the second year of the project we investigated two collision avoidance sensors for MAVs: optic flow sensors and a simple laser ranger. The following sections describe these sensors and their associated collision avoidance algorithm. Flight results are described in Section 2.4.

2.3.1 System Architecture

As shown in figure 33, the system architecture consists of five basic parts: the Virtual Cockpit, autopilot, collision avoidance sensor, simulator, and virtual sensors. The Virtual Cockpit is located on the ground station and sends waypoints to the MAV together with other commands through a 900 MHz RF link. Its main objective is to monitor and collect telemetry from the MAV. The advantage of the virtual cockpit is that it can connect to either an autopilot or a simulator with an identical interface, which makes simulation similar to flight tests. Simulation data can be collected and analyzed in the same way as flight data.

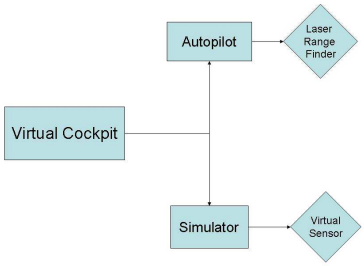


Figure 33: MAV system architecture used for simulation and flight. The Virtual Cockpit monitors flight. The autopilot is located on the aircraft and the simulator is located on a computer through a network connection.

The Autopilot is located on the MAV. It processes the control loops of the MAV and monitors its telemetry and position. The obstacle avoidance reactive planner is run on the autopilot. The autopilot connects to the Virtual Cockpit through the RF link and sends telemetry and position data to the ground station. The progress of the MAV can be monitored from the ground from the Virtual Cockpit. The autopilot runs the reactive planner obstacle avoidance algorithms and queries the laser ranger. In this work we have used the Kestrel autopilot [20] which was developed at BYU.

The simulator provides a method of testing obstacle avoidance without flying the MAV. The simulator is set up to run autopilot code in a simulated flying environment. It takes into account the dynamics of the MAV as well as environmental conditions such as wind. The result is a realistic simulation that closely models the real world. The simulator Aviones was written and developed at BYU and can be downloaded free of charge at http://sourceforge.net.

The simulator contains virtual sensors, i.e., sensors that mimic the response of real sensors on the MAV. In this case a laser ranger is one of the virtual sensors. The autopilot requests distance estimates and the simulator uses the virtual sensors to find the distance to objects in the simulated environment.

2.3.2 Collision Avoidance Using Optic Flow Sensors

Development of robust collision avoidance algorithms for fixed-wing MAVs requires a sensor for estimating distance to objects with acceptably small size and weight and high reliability. Recently, much of the research in this area has focused on utilizing vision processing techniques to estimate height-above-ground. Reference [21] demonstrated that mimicking the landing behavior of bees by maintaining constant optic flow during a landing maneuver, could be used to successfully control the descent of a MAV. Development of lightweight sensors for measurement of optic flow [22, 23, 24] has further bolstered the optic flow based approach. Reference [25] have demonstrated that these sensors can be successfully used to follow terrain in low flying MAVs.

As part of the STTR, we developed optic flow sensors and effectively demonstrated their use for autonomous landing, terrain following, and canyon following. In the following, we will describe the development of these sensors in some detail. The discussion will focus on measuring height-above-ground but the principles are identical for other distance measurements.

2.3.2.1 Height Above Ground for Autonomous Landing Optic flow sensors can be used to determine distance by relating the flow of features across an imaging array to the speed the imaging array is moving past the surface it is imaging and its distance from that surface. The number of pixels that a given object moves in the imaging plane can be combined with data from the MAV's IMU and GPS to determine distance according to

$$h = \frac{\delta x}{2\tan\left(\frac{\gamma_{\text{fov}}}{2p_n} - \frac{\dot{\theta}T_s}{2}\right)}\cos\theta\cos\phi, \tag{28}$$

where h represents height-above-ground (HAG), δx represents the displacement of the sensor parallel to the plane being imaged over the sample period T_s, γ represents the average number of pixels of displacement of features across the imaging plane over the same sample time, fov represents the field of view of the sensor, p_n is the number of pixels in the imaging array in the direction of motion of the sensor, $\dot{\theta}$ is the average pitch rate of the sensor over the sample period, θ is the average pitch of the sensor over the sample period, and ϕ is the average roll of the sensor over the sample period.

Sensor

Testing of autonomous landing using optic flow to determine HAG was performed using Agilent's ADNS-2610 optic flow sensor. The ADNS-2610 has a small form factor, measuring only 10 mm by 12.5 mm and runs at 1500 frames per second. It requires a light intensity of at least 80 mW/m^2 at a wavelength of 639 nm or 100 mW/m^2 at a wavelength of 875 nm. The ADNS-2610 measures the flow of features across an 18 by 18 pixel CMOS imager. It outputs two values, dx and dy, representing the total optic flow across the sensor's field of view in both the x and y directions. These values are stored in a buffer which can store accumulated optic flow up to 128 pixels. Once the buffer is full no more optic flow readings are accumulated until the buffer has been read. Reading the buffer clears it and allows it to begin reaccumulating optic flow measurements. The flow data in the camera y direction corresponds to lateral motion of the MAV and can be ignored when estimating HAG. The flow data in the camera x direction can be combined with data from the IMU and GPS to determine HAG according to Equation (28).

Optics A critical consideration in outfitting a MAV with an optic flow sensor is the setup of the optics. Narrow angle lenses increase the distance at which the MAV will be able to pick up appreciable optic flow. This increases the distance at which the sensor will be able to accurately measure HAG for a given groundspeed. This comes at the expense

of increasing the length of the sensor. Narrow angle lenses can also decrease the low end range of the sensor by causing overflow in the optic flow sensor's dx register or by causing the flow rate seen by the sensor to become too fast for the sensor to register. Wide angle lenses allow for smaller longitudinal sensor size, but require that larger features be available in the environment. They also decrease the distance at which appreciable optic flow will be recorded for a given groundspeed. A particular advantage of wider angle lenses is that they are less susceptible to noise introduced by angular oscillations of the MAV in pitch and roll.

Another important consideration when designing the sensor optics is lens diameter. This is especially important with regard to the corresponding f-stop value of the lens. The f-stop is defined as the ratio of the lens focal length to its diameter and is a measure of the amount of light that the optics allow to pass through to the imaging array. Lenses with larger f-stops allow less light to pass through. Keeping in mind that the imaging array requires a light intensity of 80 to 100 W/m^2 in the correct spectral range, it is important to select a lens with a low f-stop in order to increase the sensor's operational range. Because the sensor automatically adjusts shutter speed to keep the light intensity at an ideal level, there is no disadvantage to selecting a lens with a lower f-stop other than increased size. Another important implication of the f-stop is that selecting a lens with an increased focal length (a narrower angle lens) will not only increase the length of the sensor, but also increase the diameter of the sensor if the f-stop is to be maintained.

For the purposes of these tests lenses were selected which yielded fields of view of 6.5, 2.5, and 1.2 degrees when mounted on the sensor. These lenses had accompanying f-stops of 2.0, 2.0, and 2.5 respectively. Each of the configurations is shown in Figure 34. Comparable results were achieved using both the 2.5 degree and 1.2 degree field-of-view setups. However, the 1.2 degree field-of-view lens offered slightly greater range while performing better over feature-poor surfaces such as asphalt. One disadvantage of the 1.2 degree field-of-view setup up was its increased susceptibility to noise caused by pitch and roll oscillations.

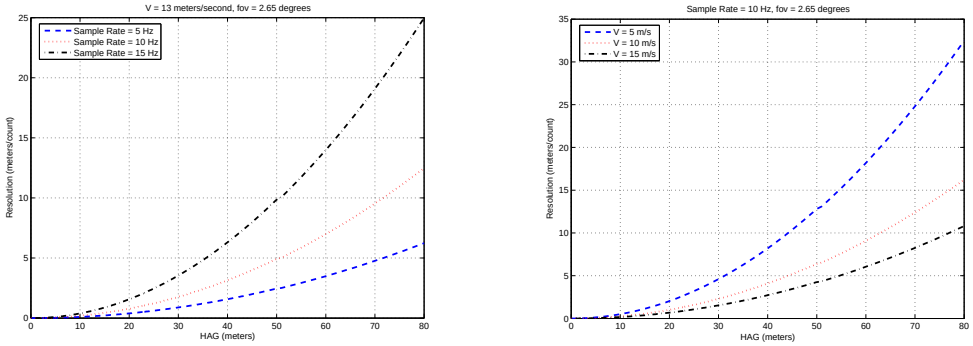
Figure 34: Optic flow sensors with three different lens configurations: 1.2, 2.5, and 6.5 degree field-of-view. The optic flow sensors are constructed by attaching a lens to an optical mouse chip.

Sample Rate Another critical consideration when equipping a MAV with an optic flow sensor is the selection of sample rate. The resolution of the sensor (measured in units of distance per sensor count) is given by

$$r = \lim_{\delta_h \to 0} \left(\frac{\delta_h \frac{\text{fov}}{2p_n}}{\tan^{-1}\left(\frac{Vt_s}{2(h-\delta_h)}\right) - \tan^{-1}\left(\frac{Vt_s}{2h}\right)} \right) \tag{29}$$

where V is groundspeed and δ_h is a perturbation in the height above ground. For a given sensor with a fixed field of view, the sensor's resolution is a function of sample rate, ground-

speed, and HAG. Plots of sensor resolution for a fixed field of view and varying sample rates and groundspeeds are shown in Figure 35.



(a) Varied sample rates.

(b) Varied groundspeeds.

Figure 35: Resolution as function of HAG.

For high values of r, each sensor count corresponds to a relatively large distance measure. Correspondingly, the noise on the sensor measured in sensor counts is amplified by r in the HAG measurement. This makes large values of r undesirable because they can significantly decrease the sensor's signal to noise ratio. It is possible to decrease r by decreasing the sample rate as shown in Figure 35(a). This, however, can be undesirable because it decreases the sensor's response time and can lead to overflow in the sensor's dx register. Furthermore, for a fixed sample rate, maintaining a low value of r for large values of HAG results in unnecessarily low values of r for lower values of HAG. This means unnecessarily low sample rates are being maintained where significantly faster sample rates could be used without significant loss in signal to noise ratio. Such an approach results in premature sensor overflow and corresponding loss of low-end range. To solve these problems two approaches were developed using a dynamic rather than static sample rate. The first approach maintains an r value along a specified resolution curve regardless of groundspeed of the MAV. The second approach uses the current velocity estimate and HAG estimate to maintain a constant value of r regardless of groundspeed and HAG.

Constant Curve Resolution In general, flight control systems for UAVs utilize a feedback loop to control around a commanded airspeed rather than groundspeed. Even when the feedback loop is closed around groundspeed, the commanded groundspeed may not be realizable due to wind conditions. In the presence of wind it is reasonable to assume that groundspeed may vary significantly depending on the heading of the MAV relative to the wind direction. As shown in Figure 35(b) variations in groundspeed can alter the resolution curve significantly. However, if the sample period is set according to

$$T_s = \frac{\tau}{V}, \tag{30}$$

where τ represents the sample rate divisor, the value of r will follow a fixed-resolution curve as a univariate function of HAG, such as those shown in Figure 35. This is useful because it eliminates the variations in resolution due to velocity and provides reliable repeatable resolutions regardless of wind conditions or commanded airspeeds.

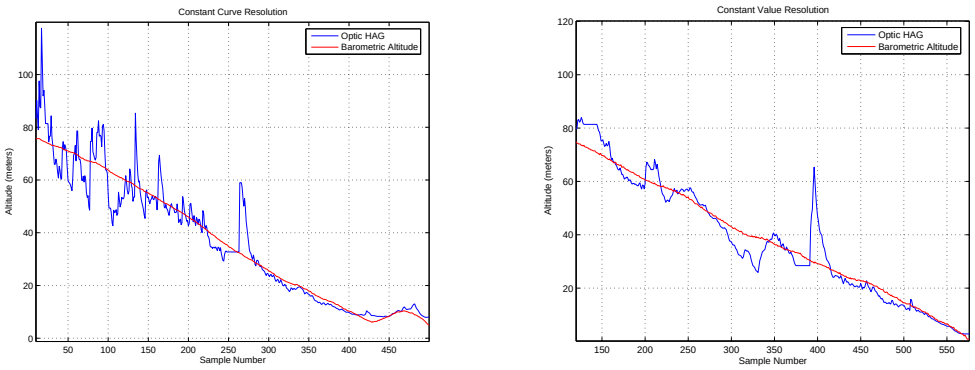
Constant Value Resolution When an estimate of the current HAG is available, the correct sample rate can be calculated so that the sensor resolution can be maintained constant regardless of HAG or groundspeed. This sample period is found by solving Equa-

tion (29) for T_s. The result is

$$T_s = \lim_{\delta_h \to 0} \left(\frac{1}{\left(\frac{V}{2(h-\delta_h)} - \frac{V}{2h}\right) \frac{1}{\tan\left(\frac{\delta_h \text{fov}}{2 p_n r}\right)} - \frac{V^2}{4h(h-\delta_h)}} \right), \tag{31}$$

where r now represents a user defined parameter that defines the resolution of the sensor for all combinations of HAG and groundspeed. In practice, the actual value of T_s is capped at both the high and low end, but allowed to vary over a given range to maintain a fixed resolution. In flight tests, using a dynamic sample rate to fix sensor resolution cut noise drastically for high values of HAG while significantly increasing the sample rate and thus improving the response time for low values of HAG.

Flight test results using both constant curve resolution and constant value resolution are shown in Figure 36. The spikes in the optic flow measurement at sample 275 in the constant curve resolution plot and 400 in the constant value resolution plot correspond to flight over an asphalt road. In practice such spikes are ignored by discarding measurements for which the optic flow sensors report poor surface quality.



(a) Fixed-curve resolution

(b) Fixed-value resolution

Figure 36: Optic HAG results from glideslope descents.

The accuracy of the distance measurements calculated from optic flow was tested by comparison with a laser rangefinder. Testing was performed by mounting the optic flow sensor and laser rangefinder perpendicular to the motion of a vehicle driving along a freeway at varying distances from the freeway sound barrier wall. Range values were recorded for both the optic flow calculations and the laser rangefinder. The results of this test are shown in Figure 37. The results show close agreement and validate the readings obtained from the optic flow calculations.

Range data from the optic flow sensor in flight tests showed reliable, repeatable readings for heights above ground less than 40 m. For the purposes of landing based on HAG calculations from optic flow, values above 40 m were discarded as noise.

2.3.2.2 Terrain Following and Canyon Navigation As small MAVs become more reliable and maneuverable, their missions will involve navigating through complex terrain, such as mountainous canyons and urban environments. In this section, we focus on terrain avoidance for flying in corridors and canyons. The algorithms we have developed enable the MAV to center itself within a corridor or canyon, or to fly near walls with a specified offset.

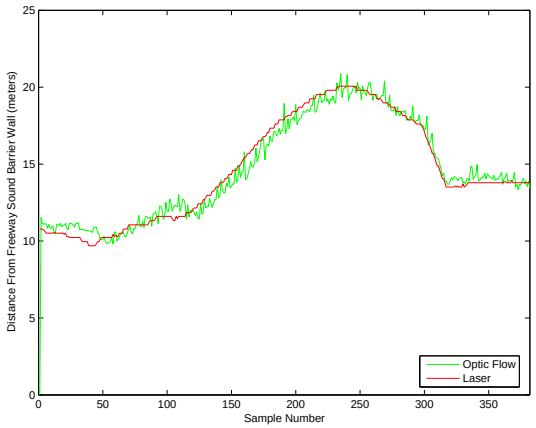


Figure 37: Optic flow range data compared to laser rangefinder data.

The algorithms utilize optic flow sensors like those shown in Figure 34. To validate our algorithms, canyon navigation flight experiments were carried out in a mountain canyon.

For terrain following and canyon navigation, the MAV was equipped with three optic-flow sensors. Two of the optic-flow sensors are forward looking but swept back from the nose by $\alpha = 60$ degrees. The third optic flow sensor points down to determine the height above ground. The optic-flow sensors, shown in Figure 34, are constructed by attaching a lens to an Agilent ADNS-2610 optical mouse sensor. The ADNS-2610 has a small form factor, measuring only 10 mm by 12.5 mm and runs at 1500 frames per second. It requires a light intensity of at least 80 mW/m^2 at a wavelength of 639 nm or 100 mW/m^2 at a wavelength of 875 nm. The ADNS-2610 measures the flow of features across an 18 by 18 pixel CMOS imager. It outputs two values, δp_x and δp_y, representing the total optic flow across the sensor's field of view in both the x and y directions. The flow data in the camera y direction corresponds to lateral motion of the MAV and is ignored.

The first step in navigating through a canyon or urban corridor is to select a suitable path through the terrain. This can be done using the RRT algorithm discussed earlier or the operator can utilize maps to define waypoints for the MAV to follow. Preplanned paths will rarely be perfect and some paths could lead the MAV near or even into uncharted obstacles. Reasons for this include inaccurate or biased terrain data, GPS error, and the existence of obstacles that have been added since the terrain was mapped. Therefore, it is important that the MAV be able to make adjustments to its path to center itself between walls and other potential hazards.

In our approach, the MAV follows its preplanned path using the vector field following method. At each time step along the path the MAV computes its lateral distance from objects to the left and right using the optic flow ranging sensors. Using this information, the MAV computes an offset δ from its planned path

$$\delta = \frac{1}{2}(D_{\text{right}} - D_{\text{left}}), \tag{32}$$

where D_{left} and D_{right} are distances to walls on the left and right measured by the optic flow sensors. Shifting the desired path by this offset centers the desired path between the detected walls as shown in Figure 38. As Figure 39 illustrates, shifting the desired path also shifts the vector field accordingly. To improve the performance of this method the optic ranging sensors are pointed forward at a 30 degree angle. This reduces lag caused by

filtering the sensor readings and allows the MAV to detect obstacles ahead of its current position.

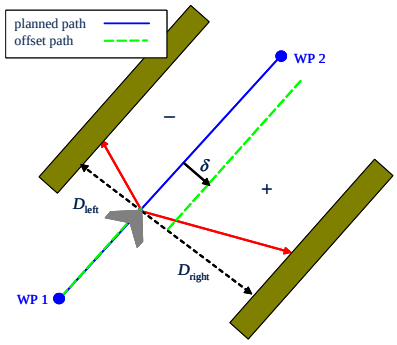


Figure 38: Using the measurements from the optic flow sensors, the planned path (solid blue) is shifted by δ to create a new desired path (dashed green) that is centered between the canyon walls.

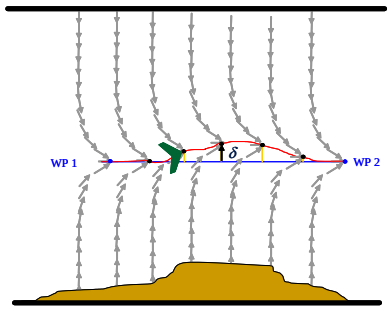


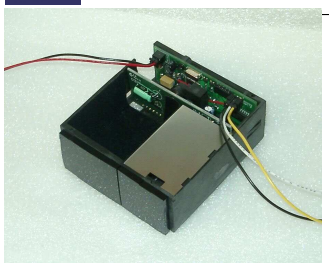
Figure 39: The adjusted path (red) is offset from the preplanned path (blue) by the calculated offset (δ) at each time step to center the desired path between the canyon walls, thus shifting the vector field along with it.

2.3.3 Collision Avoidance Using a Laser Range Finder

The laser ranger is a small low power device (approximately 2 Watts) that can be mounted on a MAV as shown in figure 40. It has a range of 400 meters on non-reflective surfaces and runs at 10 Hz [26]. It was mounted to detect obstacles directly in front of the MAV. In hardware we did not use a scanning algorithm for the laser. However, in simulation we assumed a laser ranger with two degrees of freedom and use the scanning algorithm.

The laser ranger is used to implement a reactive path planning scheme that causes the MAV to deviate from the original waypoint path if obstacles are detected along that path.

Consider the pop-up threat scenario shown in Figure 41. Figure 41(a) shows the starting position of the MAV as it enters a situation where obstacle avoidance is required. We assume the MAV has a forward ground velocity and is trying to track the given waypoint path. We also assume the laser ranger has detected enough points on objects to safely avoid the obstacle. The MAV detects the obstacles in figure 41(b). Notice that not all of the points are in front of the MAV. These points represent past obstacles that the laser ranger



(a) Laser ranger from Opti-Logic

(b) Laser ranger mounted on a MAV

(c) MAV with Laser Ranger

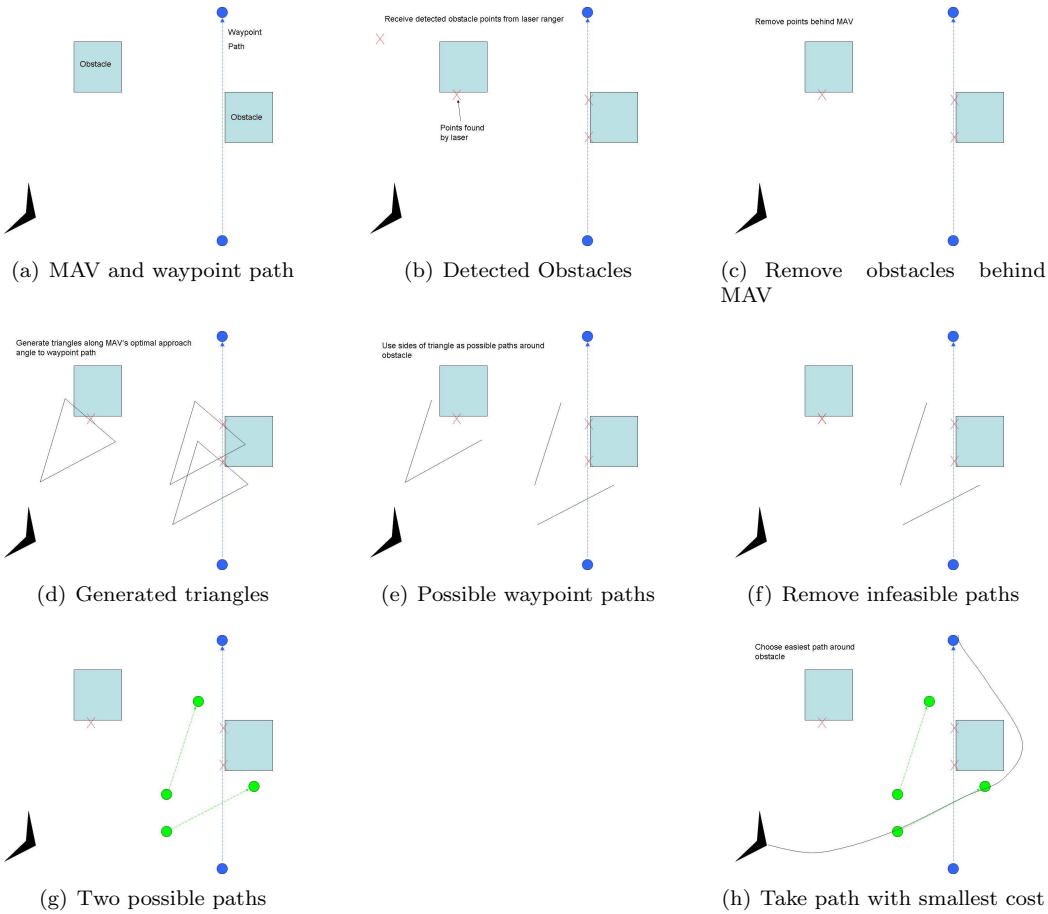
Figure 40: Photographs show Opti-Logic's laser ranger mounted on a MAV ready to fly the reactive planner obstacle avoidance algorithm.

detected but are no longer relevant. These must be removed as in figure 41(c). To avoid the obstacles, we propose generating avoidance triangles around the obstacles and using the sides of the triangles as possible waypoint paths. Since the dimensions of the obstacles are unknown, they must be estimated by construction of a triangle around each point, as shown in Figure 41(d). Notice that some of the triangle sides intersect. If sides intersect, then they are not feasible paths around the obstacles. Figure 41(e) removes the intersecting sides. The remaining sides are all possible waypoint paths around the detected obstacles.

The MAV cannot always track each path generated before hitting an obstacle. In addition, some paths can be tracked faster than others. Using an estimation method described later, we can estimate at what point $\mathbf{r}$ the MAV can track each path. Using $\mathbf{r}$, we can determine whether the path can be tracked before hitting the obstacle. Infeasible paths can be removed as in figure 41(f). The remaining paths can be assigned a cost by using the distance from the MAV to $\mathbf{r}$, and the a path selected based on this cost. The final selected path is shown in figure 41(h)

There are a few assumptions made in the reactive obstacle avoidance algorithm. First, we assume that the laser range finder has found enough points on pending obstacles so that the inner loop can plan a path around them. This implies that the inner loop and laser range finder scanning algorithm work together to provide sufficient data for obstacles. Second, we assume that the autopilot can quickly track desired headings. There is some room for error in this as will be seen below.

The steps to avoiding obstacles by generating intermediate waypoint paths make use of the limit cycle waypoint trajectory tracking method described in Ref. [**?**]. The first step is to calculate the desired approach angle (ψ_d) to the current waypoint path which gives the heading the MAV needs to track to the obstacle avoidance algorithm. Using ψ_d, the MAV can always progress towards the waypoint path while still avoiding obstacles. The second step removes all points behind the MAV and farther than $1.25 * triangle\_length$ from the MAV. The reason for removing all points over a certain distance from the UAV is to prevent unneeded processing in deciding which path to follow. The third step is triangle generation. Given that the MAV cannot detect the exact size of the obstacle, the algorithm makes a buffer region around the point using a triangle. The triangle must be isosceles and the angle bisector of the odd angle must have the same heading as ψ_d as shown in figure 42(a). Make a triangle for each point. The result will be multiple triangles that may overlap, but are all of equal size as in figure 42(b) Notice that *triangle_width* and *triangle_length* are two important dimensions of the triangles (figure 42(c)). *triangle_length* is the distance the MAV has to maneuver around the obstacle. If the MAV has a large turning radius, then *triangle_length* will have to be large as well. *triangle_width* is the buffer region between the obstacle point and the new waypoint path. As expected obstacle size grows, so will



(a) MAV and waypoint path

(b) Detected Obstacles

(c) Remove obstacles behind MAV

(d) Generated triangles

(e) Possible waypoint paths

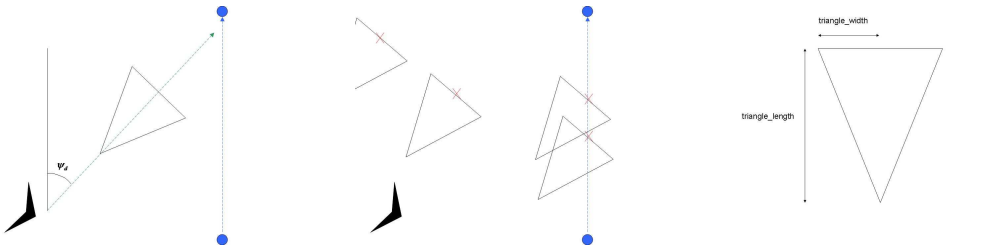
(f) Remove infeasible paths

(g) Two possible paths

(h) Take path with smallest cost

Figure 41: Reactive planner example. The MAV must detect and avoid obstacles while approaching a pre-planned waypoint path.

triangle_width.



(a) Optimal Approach Angle to waypoint path(ψ_d).

(b) Generate triangles around points.

(c) Triangle dimensions.

Figure 42: Triangle generation algorithm . Triangle sides are used as possible waypoint paths around obstacles.

The next step is to remove intersecting sides. There will be at least two remaining sides after the intersection elimination. This is proven by the fact that each equivalent side is parallel. No matter how the triangles are arranged, at least a set of non-parallel sides will remain. Because they do not intersect any other paths, and by our assumptions that we know all the points around an obstacle, then those paths also avoid obstacles.

There are two or more remaining triangle sides that can now be used as waypoint paths around the obstacle. The remaining task is to choose the best path. Each path is fairly close to the MAV and deviate equally from ψ_d. Unfortunately, there will still be some paths that cannot be tracked before colliding with an obstacle. We need a cost function that estimates how fast the MAV can track each path. A good cost function is simply the distance of the MAV to the first point on the waypoint path.

There are two cases to consider to estimate the point by which the MAV can track the waypoint path. First, determine whether the MAV needs to turn left or right to track the path. This can be done by calculating the cross product of the position vector of the MAV by the waypoint being approached minus the MAV position vector (e.g. $\mathbf{u} \times (\mathbf{w_2} - \mathbf{u})$). If the cross product is positive, the MAV must turn left, otherwise right. Now create a circle to the left or right of the MAV as needed using the turning radius. The circle should be positioned to represent the turn of the MAV. Translate the circle and waypoint path such that the center of the circle is at the origin. This simplifies the calculations. We must find where the circle intersects the waypoint path. First take the equation of the waypoint path:

$$y_i = m(x_i - u_x) + u_y \tag{33}$$

where x_i and y_i are the intersection points of the waypoint path and circle and m is the slope. The equation of the circle is:

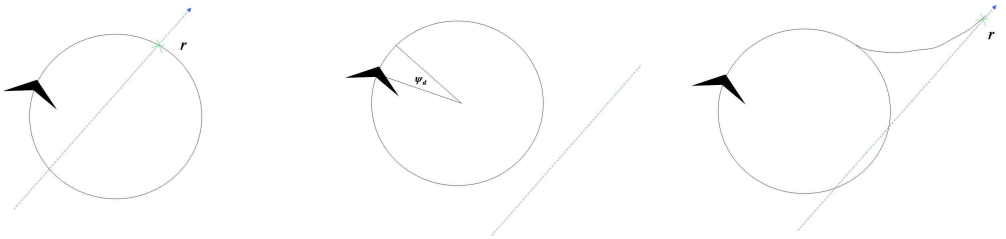
$$r_t^2 = x_i^2 + y_i^2. \tag{34}$$

Substituting and solving for x_i we get

$$x_i = \frac{1}{2(1+m^2)}(2m^2 w_{2x} - 2m w_{2y} + 2\sqrt{-m^2 w_{2x}^2 + 2m w_{2x} w_{2y} - w_{2y}^2 + r_t + m^2 r_t}) \tag{35}$$

y_i can be obtained by substituting x_i back into equation (33). In the case that the square root is negative, the circle and waypoint path do not intersect and we know that we must use case two. If it is not negative, then we must find where x_i and y_i are on the circle. Translate the waypoint path such that it crosses through the center of the circle. Take the

cross product of the waypoint path and x_i and y_i. If the cross product is negative, use case two. Otherwise use case one. Case one consists of using x_i and y_i as the coordinate $\mathbf{r}$ which indicates the position at which the MAV will track the waypoint path as indicated in figure 43(a).



(a) Case 1. (b) ψ_d. (c) Case 2.

Figure 43: Waypoint path interception.

Case two projects the MAV position onto the waypoint path and adds an estimated distance. The estimated distance is calculated by finding ψ_d, the angle of the circle across the distance needed to fly before flying parallel to the waypoint path (figure 43(b)). That distance plus another two turning radii along the waypoint path from the projection are the estimated distance to $\mathbf{r}$, the point of tracking the waypoint path. This is illustrated in figure 43(c).

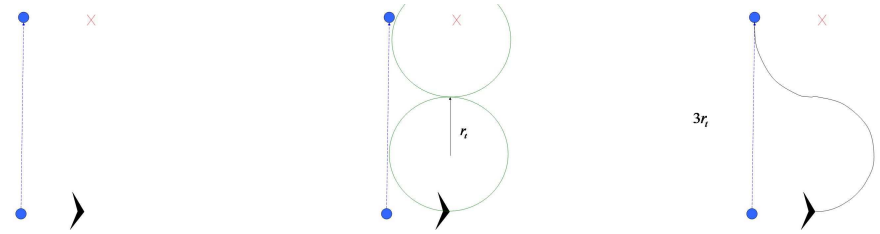
The cost function is the distance from the MAV to $\mathbf{r}$. The smallest cost is the path that can be tracked in the shortest distance. Choose the path with the smallest cost as the new waypoint path.

The reactive path planner is summarized by the following steps.

1. Calculate ψ_d.
2. Remove all points behind the MAV and all points over $1.25 * triangle\_length$ distance away from the MAV.
3. Generate triangles with dimensions $triangle\_width$ and $triangle\_length$ along the angle ψ_d.
4. Remove intersecting sides of triangles. The remaining sides are possible waypoint paths.
5. Calculate the estimated point of tracking for each of the waypoint paths ($\mathbf{r}$). The distance to this point is the cost for the corresponding waypoint path.
6. Choose the path with the smallest cost as the new waypoint path.

Analysis This section presents conditions under which the UAV is guarantees to track a generated path from the reactive planner. The constraints of a fixed wing air vehicle and limited processing efficiency severely limit finding feasible paths in all situations, but under certain conditions, successful avoidance can be guaranteed.

Consider figure 44(a). The red X represents a detected obstacle and the blue dashed line is the desired waypoint path to track. The MAV is angled such that the obstacle is slightly in front, thus not filtering out the obstacle. To avoid the obstacle, the MAV must turn nearly 180 degrees toward the waypoint path and then turn back 90 degrees to begin tracking the path. This can be considered a worst case scenario. Any other scenario requires less distance to track the path.



(a) Worse case senerio (b) Turning Radius (c) Quickest path

Figure 44: Analysis

Theorem 2.4 *Given a MAV with turning radius r_t, parameter triangle_length $\geq 3r_t$, and assuming (1) no wind, (2) sufficient points have been detected on a given obstacle, and (3) that the MAV is triangle_length distance away from the obstacle, the MAV can successfully track the waypoint path generated around that obstacle.*

Proof: Consider the situation where the MAV approaching a set of obstacles that requires it to track a path requiring at least a 90 degree heading change as depicted in figure 44(b). To track the path, the MAV must turn 180 degrees toward the path, and then turn another 90 degrees to track the path, as shown in figure 44(c). The MAV may cover up to $3r_t$ distance before tracking the path around the obstacle. The parameter *triangle_length* is the distance from an obstacle where a path begins. Therefore, if the UAV is at least $3r_t$ away from the obstacle, the MAV is guaranteed to avoid it. ■

Simulation Results

The simulator is the primary testing method for our architecture making it an important aspect in understanding the results. The simulator generates random cities through which the MAV can be flown. The parameters that can be changed are street width, building width, mean building height, and building height variance. The buildings created are placed on a flat terrain and can be detected by a simulated laser ranger. The buildings are rectangular and protrude vertically from the flat terrain surface.

The physics engine of the simulator models aerodynamic forces, gravitational forces, forces due to control surfaces, forces due to thrust, and forces due to wind. The MAV aerodynamic coefficients have been tuned to match the airframes used in our hardware testbed.

The simulator also emulates the Kestrel autopilot used at BYU [20]. It receives a .dll file of compiled Kestrel autopilot code and runs it at real time in an attempt to match the speed and timing of the MAV autopilot. The controls and sensors are run by the autopilot in the simulator in the sense that they are executed in hardware as algorithms. This method makes the simulation a close match to actual flight tests facilitating rapid prototyping.

Avoiding a single obstacle allows the algorithm parameters like *triangle_width* and *triangle_length* to be appropriately tuned. The parameters can be adjusted based on convergence speed to the desired waypoint, desired distance from obstacle during avoidance, and desired distance from obstacle to begin avoidance. In the simulation depicted in figure 45, the MAV was directed to fly on a waypoint path directly through a building. The building is represented as a solid gray box in the image with a width of 20 meters. The solid white line is the waypoint path. The dotted line is the actual path flown by the MAV. The

parameters of the algorithm are:

$$\begin{aligned} \text{turning\_radius} &= 30 \\ \text{triangle\_length} &= 3 \cdot \text{turning\_radius} \\ \text{triangle\_width} &= 35. \end{aligned}$$

The MAV used the virtual laser range finder running at three Hertz to detect the position of the obstacle and effectively avoided it.

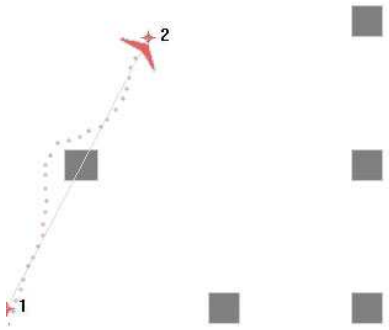


Figure 45: Single obstacle simulation

2.4 Objective 3: Hardware Demonstration

2.4.1 Experimental Platform

BYU has developed a reliable and robust platform for testing unmanned air vehicles [20, 27, 28]. Figures 46 through 48 show the key elements of the testbed. Figure 46 shows the Kestrel autopilot which is equipped with a Rabbit 3400 29 MHz processor, rate gyros, accelerometers, absolute and differential pressure sensors. The autopilot measures 3.8 × 5.1 × 1.9 cm and weighs 17 grams.

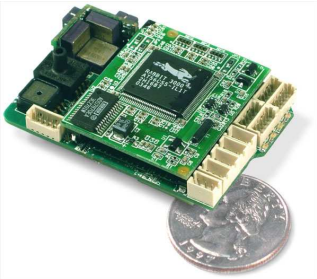
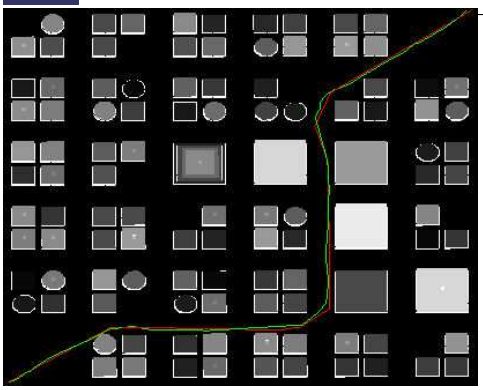


Figure 46: The Kestrel autopilot system developed at BYU.

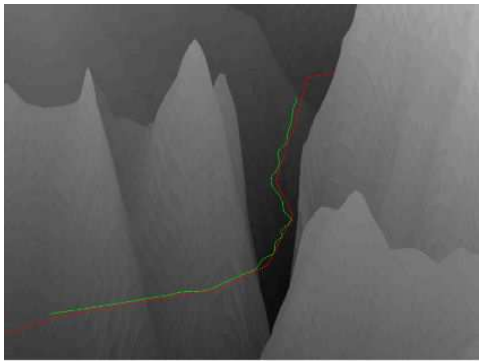
Figure 47: Zagi foam flying wing airframe.

Figure 48: Screen shot of the Virtual Cockpit software used to interact with the MAV.

Figure 47 shows on of the airframes used for the flight tests described in this report. The airframe is a 1.2 meter wingspan Zagi XS EPP foam flying wing, which was selected for



(a) Flight test data through synthetic urban terrain.



(b) Flight test data through canyon in southern Utah.

Figure 49: Flight tests of the RRT algorithm (red = planned waypoint path, green = actual telemetry plot).

its durability, ease of component installation, and flight characteristics. Embedded in the airframe are the Kestrel autopilot, batteries, a 1000 mW, 900 MHz radio modem, a GPS receiver, a video transmitter, and a small analog camera.

Figure 48 shows a screen shot of the Virtual Cockpit software that interfaces through a communication box to the MAVs. Virtual Cockpit allows the user to control the behavior of the MAV by changing control gains, specifying waypoints, or changing experimental objectives. Telemetry from the MAV is displayed in the Virtual Cockpit in real time allowing the user to monitor the progress of the experiment.

2.4.2 Flight Test Results for Waypoint Path Planning

Although we have not attempted to fly our UAVs through a real city to verify these results, we have done some other tests that indicate how effective this planning method would be in a true city. We used the RRT path planner to plan a constant-altitude path through a virtual city with 30m-wide streets. To test this path, we sent the resultant waypoints to a UAV flying over flat terrain. We commanded the UAV to follow this path using the path following method expected by the planner. We then superimposed the telemetry data from the UAV on the synthetic urban terrain, along with the planned waypoint path. From this plot (Figure 49(a)) we see the UAV was able to fly through this virtual city without any collisions with the buildings. Similar results were obtained when the UAV flew different paths through the city. Although the UAV was not flying in wind conditions that accurately represent urban terrain, it was compensating for open space wind conditions, where wind speeds were roughly 30% of the airspeed.

We have also imported USGS terrain data (10m precision) into the RRT path planner to use in finding paths through real terrain. Using this data, we planned a path with GPS waypoints for the UAV to follow through a narrow canyon in Central Utah. The path planner quickly found a path free from collision with the terrain. The UAV successfully followed this path through the canyon while commanding a constant height above ground. Flight results are shown in Figure 49(b).

2.4.3 Flight Test Results for Waypoint and Orbit Following

To demonstrate the path following abilities enabled by the vector field algorithm, MAVs were commanded to fly a variety of paths composed of straight lines, orbits, and combinations

of straight lines and circular arcs. The results presented were gathered on a relatively calm day with the average wind speed measuring 0.9 m/s from 230 degrees southwest. Gusts were as high as 2.5 m/s. The commanded airspeed was 13 m/s, thus for these tests the average wind speed and wind gusts were respectively 7 percent and 20 percent of the commanded airspeed.

To illustrate orbit following with the vector field algorithm, the MAV was commanded to fly a series of concentric orbits with varying radii. The results are shown in Figure 50. The mean lateral path error for the 150 m orbit was 0.58 m, while the standard deviation of the path error was 0.42 m. For the 100 m orbit the mean path error was 0.65 m and the standard deviation of the path error was 0.46 m. For the 70 m orbit, the mean and standard deviation of the lateral error was 1.9 m and 1.1 m respectively. For the larger orbits, the mean errors are about half of the wingspan of the MAV. The data indicate that tight tracking is more difficult to achieve as the orbit radius decreases. This is expected since control adjustments for loss in altitude in turns must be mixed with those used to control the lateral error and since the states associated with a smaller orbit are further away from the nominal wings-level trim condition.

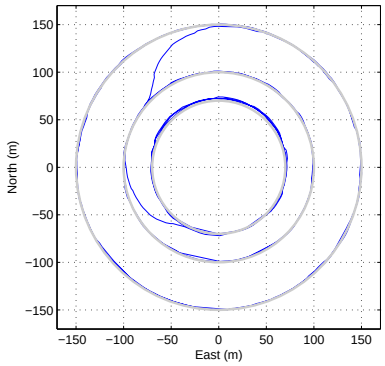


Figure 50: Telemetry plot for orbits with radii of 150, 100, and 70 m.

Figure 51 illustrates the ability of the MAV to follow straight line segments with acute angles. Excluding the transient errors from the turns, the mean following error on the straight-line portions of the path was 0.74 m with a standard deviation of 0.66 m.

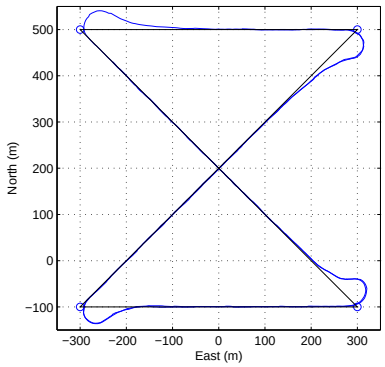


Figure 51: Telemetry plot for straight line following.

A combination of the straight-line and arc-following methods was also tested. The techniques described in Section 2.2.2.3 were implemented and the results are plotted in Figure 52. The thicker line represents the desired path that was planned to equalize the straight-line and smoothed path lengths. The mean path following error and standard deviation were 3.6 m and 5.1 m respectively. Although the transitions from the straight line to the orbit portions show some lateral following errors, the length of the path flown and the desired length are similar. The length of the straight-line path shown in Figure 52 was 2897 m. The actual distance flown was 27 m less than the desired distance, which is an error of only 0.93 percent.

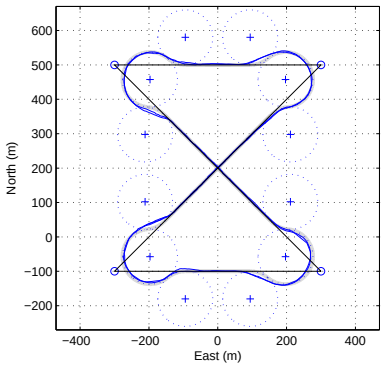


Figure 52: Telemetry plot for equal path length following.

To further test the robustness and capabilities of the proposed path-following algorithms, many other types of paths have been flown. The path shown in Figure 53 illustrates both obtuse and acute angles and the decision of the path follower to cut the corners of the obtuse angles and flare out and around on the acute angles. Considering the path following error over the full path gave a mean error of 3.6 m and a standard deviation of 4.7 m. Figure 54 shows a path representative of an urban scenario. Although these are actual flight results, the terrain is synthetic. The straight-line follower was used to follow this path. The mean lateral error over the full path was 3.4 m, while the standard deviation was 4.8 m.

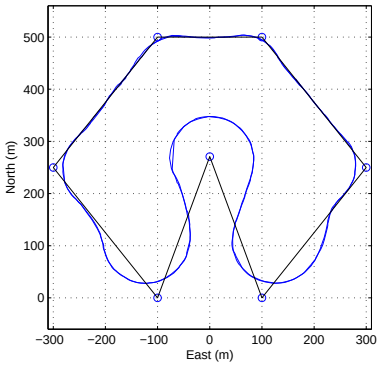


Figure 53: Combination of equal path length and corner cutting following.

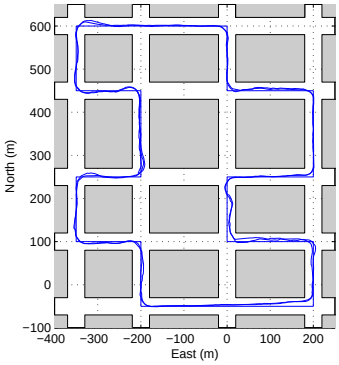


Figure 54: Urban terrain following using straight line following.

2.4.4 Flight Test Results using Optic Flow Sensors

Goshen Canyon in central Utah was chosen as a flight test site. This canyon was selected for its steep winding canyon walls that reach over 75 m in height, as well as its proximity to BYU and low utilization. Flight tests through Goshen Canyon were conducted using the fixed-wing MAV. Photographs of the flight tests taken by observers and the onboard camera are shown in Figure 55. In the first flight through the canyon, the planned path was selected to follow the road. The MAV navigated the canyon with only minor adjustments to its path. For the second flight, the planned path was intentionally biased into the east canyon wall to verify that the navigation algorithms would correct the planned path toward the center of the canyon, enabling the MAV to avoid the canyon walls.

Figure 55: This figure shows the MAV as it enters Goshen Canyon. The inset is an image from the camera on-board the MAV.

Figures 56 shows results from the second flight which demonstrate that the MAV biased its desired path up to 10 m to the right to avoid the canyon walls. If the MAV had not biased its path it would have crashed into the east canyon wall.

2.4.5 Flight Test Results using Laser Range Finder

A successful test of the reactive planner required at least one obstacle high enough to fly the MAV at without a need to change altitude low to the ground. This prevents the laser ranger from detecting points on the ground that might be mistakenly considered obstacles and allows for short altitude drops during turns.

We chose an isolated building on the BYU campus that is 50 meters high and 35 meters square. Other surrounding buildings rose to about 20 meters which separated the building as a single obstacle. We planned a path from the south side of the building to the north at an altitude of 40 meters. The MAV had no previous information on the location and dimensions of the building. The turning radius of the MAV is approximately $r_t = 30$ meters. We set $triangle\_length = 90$ meters and $triangle\_width = 55$ meters to ensure that one detected point on the building generated a path sufficiently to the side of the building. A GPS telemetry plot of the results is shown in figure 57.

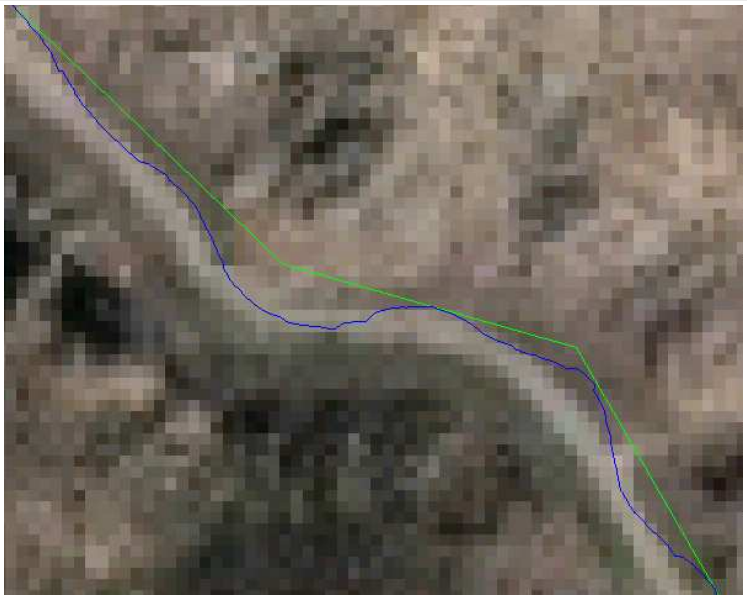


Figure 56: Results from the second flight through Goshen Canyon. Flight test results show the planned path (green) and the actual path (blue). The planned path was intentionally biased to the east forcing the MAV to offset from its planned path to center itself through the canyon.

As the MAV approached the building, the laser ranger detected the building and calculated its position. When the MAV came within 90 meters of the building, the reactive planner generated a path around the building and the MAV began to track the path. Notice that as the MAV passed the building, it once again began to track the original waypoint path that went through the building. The MAV successfully avoided the building without human intervention. Figure 58 shows images of the MAV and its camera view as it executed the avoidance maneuver.

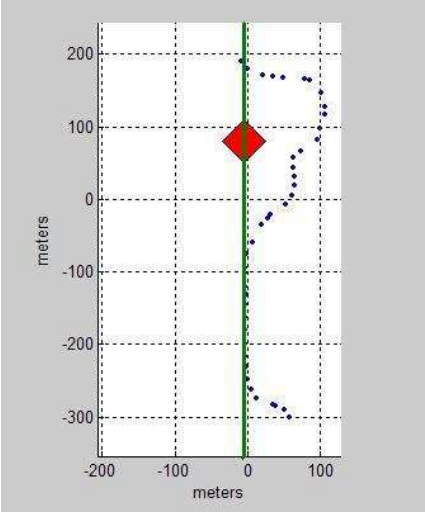


Figure 57: Flight test results plot of a MAV avoiding a building with a planned path through the building.

Figure 58: In-flight image of the Kimball Tower on BYU campus during the collision avoidance maneuver.

3 Technology Transitions

This STTR effort has directly resulted in several technology transfer opportunities. Successful transition of technology to activities of current and practical interest to the Air Force has occurred. For example:

- **AFRL/MN.** Efforts to fly trajectories on UAV hardware led to work with AFRL/MN to develop miniature autopilot hardware and software technologies that were recently used by Air Force Special Operations forces in war games in Mississippi.
- **NASA/Ames.** The waypoint path planning algorithms and trajectory generation methods developed during the STTR has been utilized by the Army/NASA Rotorcraft Division in their Precision Autonomous Landing Adaptive Control Experiment (PALACE) program.
- **Auto-pilot technology.** The autopilot developed in part under the STTR and used extensively in Phase II for testing and demonstration has generated enough interest by university and government laboratory researchers that a small company, Procerus, was organized to manufacture and market autopilot hardware and software.

During the STTR, efforts were made by ISL and BYU to make the major prime contractors aware of the technology developed for autonomous guidance and control and its potential benefits for their current and future programs. Partnership with Raytheon is being pursued to transition the technology to benefit Air Force programs and to continue the development of the technology in applications of interest to the Air Force.

4 Recommendations for Further Work

During Phase II, robust algorithms for path planning and trajectory generation were implemented and tested on micro-air vehicles. The next step is to transition and incorporate the software developed into emerging UAV applications. Further demonstrations of the technology developed in Phase II will be used to build interest in teaming arrangements for follow-on Phase III work in conjunction with auto-pilot and sensor manufacturers.

ISL will continue to pursue two avenues of commercialization. The first is to partner with Raytheon to develop autonomous route-planning software with applications to future military UAV platforms. The second avenue is to market the collision avoidance technology developed in Phase II and/or partner with companies developing sensor technology for UAVs. Beyond military applications, the autonomous guidance and control algorithms demonstrated in Phase II have application to border and police surveillance in urban areas, and hazardous waste and environmental monitoring.

5 List of Key Personnel Supported

ISL Personnel: Michael L. Larsen, Carolyn Christensen

BYU Faculty: Randal W. Beard, Timothy W. McLain

BYU Students: Derek Kingston, Joshua Hintze, Stephen Griffiths, Jeff Saunders, Andrew Curtis, Carolyn Christiansen, Blake Barber, Wei Ren, Brandon Call, Josh Redding,

6 List of Publications

1. Stephen Griffiths, Jeff Saunders, Andrew Curtis, Tim McLain, Randal Beard, "Obstacle and Terrain Avoidance for Miniature Aerial Vehicles," *IEEE Robotics and Automation Magazine*, (to appear).

2. Derek R. Nelson, Blake Barber, Timothy W. McLain, Randal W. Beard, "Vector Field Path Following for Miniature Air Vehicles," *IEEE Transactions on Robotics*, (to appear).

3. Randal Beard, Derek Kingston, Morgan Quigley, Deryl Snyder, Reed Christiansen, Walt Johnson, Timothy Mclain, Mike Goodrich, "Autonomous Vehicle Technologies for Small Fixed Wing UAVs," *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 1, January, 2005, pp. 92–108.

4. Erik P. Anderson, Randal W. Beard, Timothy W. McLain, "Real Time Dynamic Trajectory Smoothing for Uninhabited Aerial Vehicles," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 3, May, 2005, pp. 471–477.

5. Wei Ren, Randal W. Beard, "Trajectory Tracking for Unmanned Air Vehicles with Velocity and Heading Rate Constraints," *IEEE Transactions on Control Systems Technology*, vol. 12, no. 5, September, 2004, pp. 706–716.

6. Brandon Call, Randal W. Beard, Clark Taylor, "Obstacle Avoidance for Unmanned Air Vehicles Using Image Feature Tracking," *AIAA Conference on Guidance, Navigation, and Control*, Keystone CO, 2006. (to appear).

7. Derek R. Nelson, D. Blake Barber, Timothy W. McLain, Randal W. Beard, "Vector Field Path Following for Small Unmanned Air Vehicles," *American Control Conference*, Minneapolis, Minnesota, June 2006, pp. 5788–5794.

8. Joshua Redding, Timothy W. McLain, Randal W. Beard, Clark Taylor, "Vision-based Target Localization from a Fixed-wing Miniature Air Vehicle," *American Control Conference*, Minneapolis, Minnesota, June 2006, pp. 2862–2867.

9. Wei Ren, Ji-Sang Sun, Randal W. Beard, Timothy W. McLain, "Nonlinear Tracking Control for Nonholonomic Mobile Robots with Input Constraints: An Experimental Study," *American Control Conference*, Portland, OR, MA, June, 2005, p. 4923–4928.

10. Morgan Quigley, Michael A. Goodrich, Steve Griffiths, Andrew Eldredge, Randal W. Beard, "Target Acquisition, Localization, and Surveillance Using a Fixed-Wing Mini-UAV and Gimbaled Camera," *IEEE International Conference on Robotics and Automation*, 2005.

11. D.J. Lee, R.W. Beard, P.C. Merrell, and P. Zhan "See and Avoidance Behaviors for Autonomous Navigation," *SPIE Optics East, Robotics Technologies and Architectures*, Mobile Robot XVII, vol. 5609–05, Philadelphia, PA USA, October 25–28, 2004.

12. P. Zhan, D.J. Lee, and R.W. Beard, "Solving Correspondence Problem Using 1-D Signal Matching," *SPIE OpticsEast, Robotics Technologies and Architectures*, Intelligent Robots and Computer Vision XXII, vol. 5608-24, Philadelphia, PA USA, October 25-28, 2004.

13. P.C. Merrell, D.J. Lee, and R.W. Beard, "Statistical Analysis of Multiple Optical Flow Values for Estimation of Unmanned Air Vehicles Height Above Ground", *SPIE OpticsEast, Robotics Technologies and Architectures*, Mobile Robot XVII, vol. 5608-28, Philadelphia, October 25-28, 2004.

14. Wei Ren, Randal W. Beard, "Constrained Nonlinear Tracking Control For Small Fixed-wing Unmanned Air Vehicles," *American Control Conference*, Boston, MA, June, 2004, p. 4663–4668.

15. Derek Kingston, Randal Beard, Timothy McLain, Michael Larsen, Wei Ren, "Autonomous Vehicle Technologies for Small Fixed Wing UAVs," *AIAA 2nd Unmanned Unlimited Systems, Technologies, and Operations–Aerospace, Land, and Sea Conference and Workshop & Exhibit,* San Diego, CA, September, 2003, Paper no. AIAA-2003-6559.

References

[1] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *International Journal of Robotic Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[2] J. B. Saunders, B. Call, A. Curtis, R. W. Beard, and T. W. McLain, "Static and dynamic obstacle avoidance in miniature air vehicles," in *AIAA Infotech@Aerospace*, no. AIAA-2005-6950. Arlington, Virginia: American Institute of Aeronautics and Astronautics, September 2005.

[3] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," October 1998, tR 98-11, Computer Science Dept., Iowa State University.

[4] J. J. Kuffner and S. M. LaValle, "RRT-connect: An efficient approach to single-query path planning," in *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, April 2000, pp. 995–1001.

[5] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," in *Algorithmic and Computational Robotics: New Directions*, B. R. Donald, K. M. Lynch, and D. Rus, Eds. Wellesley, MA: A. K. Peters, 2001, pp. 293–308.

[6] E. P. Anderson, "Constrained extremal trajectories and unmanned air vehicle trajectory generation," Master's thesis, Brigham Young University, Provo, Utah 84602, April 2002, available at http://www.ee.byu.edu/magicc/publications/thesis/ErikAnderson.ps.

[7] E. P. Anderson, R. W. Beard, and T. W. McLain, "Real time dynamic trajectory smoothing for uninhabited aerial vehicles," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 3, pp. 471–477, May 2005.

[8] D. B. Kingston and R. W. Beard, "Real-time attitude and position estimation for small UAVs using low-cost sensors," in *AIAA 3rd Unmanned Unlimited Technical Conference, Workshop and Exhibit*, Chicago, IL, September 2004, aIAA Paper No. 2004-6488 (to appear).

[9] S. Park, J. Deyst, and J. How, "A new nonlinear guidance logic for trajectory tracking," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, August 2004, AIAA-2004-4900.

[10] I. Kaminer, A. Pascoal, E. Hallberg, and C. Silvestre, "Trajectory tracking for autonomous vehicles: An integrated approach to guidance and control," *AIAA Journal of Guidance, Control and Dynamics*, vol. 21, no. 1, pp. 29–38, 1998.

[11] P. Aguiar, D. Dačić, J. Hespanha, and P. Kokotivić, "Path-following or reference-tracking? An answer relaxing the limits to performance," in *Proceedings of IAV2004, 5th IFAC/EURON Symposium on Intelligent Autonomous Vehicles*, Lisbon, Portugal, 2004.

[12] A. P. Aguiar, J. P. Hespanha, and P. V. Kokotovic, "Path-following for nonminimum phase systems removes performance limitations," *IEEE Transactions on Automatic Control*, vol. 50, no. 2, pp. 234–238, February 2005.

[13] J. Hauser and R. Hindman, "Maneuver regulation from trajectory tracking: Feedback linearizable systems," in *Proceedings of the IFAC Symposium on Nonlinear Control Systems Design*, Tahoe City, CA, June 1995, pp. 595–600.

[14] P. Encarnação and A. Pascoal, "Combined trajectory tracking and path following: An application to the coordinated control of marine craft," in *Proceedings of the IEEE Conference on Decision and Control*, Orlando, FL, 2001, pp. 964–969.

[15] R. Skjetne, T. Fossen, and P. Kokotović, "Robust output maneuvering for a class of nonlinear systems," *Automatica*, vol. 40, pp. 373–383, 2004.

[16] R. Rysdyk, "UAV path following for constant line-of-sight," in *Proceedings of the AIAA 2nd Unmanned Unlimited Conference.* AIAA, September 2003, paper no. AIAA-2003-6626.

[17] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, 1985, pp. 500–505.

[18] K. Sigurd and J. P. How, "UAV trajectory design using total field collision avoidance," in *Proceedings of the AIAA Guidance, Navigation and Control Conference*, August 2003.

[19] H. K. Khalil, *Nonlinear Systems*, 3rd ed. Upper Saddle River, NJ: Prentice Hall, 2002.

[20] "Procerus technologies," http://procerusuav.com/.

[21] J. Chahl, M. Srinivasan, and S. Zhang, "Landing strategies in honeybees and applications to uninhabited airborne vehicles," *The International Journal of Robotics Research*, vol. 23, no. 2, pp. 101–110, 2004.

[22] G. Barrows and C. Neely, "Mixed-mode VLSI optic flow sensors for in-flight control of a micro air vehicle," in *Proceedings SPIE*, San Diego, August 2000, pp. 52–63.

[23] F. Ruffier and N. Franceschini, "Visually guided micro-aerial vehicle: automatic take off, terrain following, landing and wind reaction," in *Proceedings of the 2004 IEEE International Conference on Robotics & Automation*, New Orleans, 2004, pp. 2339–2346.

[24] J.-C. Zufferey and D. Floreano, "Toward 30-gram autonomous indoor aircraft: Vision-based obstacle avoidance and altitude control," in *Proceedings of the 2005 IEEE International Conference on Robotics & Automation*, Barcelona, April 2005.

[25] G. L. Barrows, J. S. Chahl, and M. V. Srinivasan, "Biomimetic visual sensing and flight control," *The Aeronautical Journal, London: The Royal Aeronautical Society*, vol. 107, pp. 159–168, 2003.

[26] http://www.opti-logic.com/industrial_rangefinders.htm.

[27] R. Beard, D. Kingston, M. Quigley, D. Snyder, R. Christiansen, W. Johnson, T. McLain, and M. Goodrich, "Autonomous vehicle technologies for small fixed wing UAVs," *AIAA Journal of Aerospace, Computing, Information, and Communication*, vol. 2, no. 1, pp. 92–108, January 2005.

[28] R. W. Beard, D. Lee, M. Quigley, S. Thakoor, and S. Zornetzer, "A new approach to observation of descent and landing of future Mars mission using bioinspired technology innovations," *AIAA Journal of Aerospace Computing, Information, and Communication*, vol. 2, no. 1, pp. 65–91, January 2005.